



**Ricardo Filipe Chaves Gaspar**

Master of Science in Computer Science

## **Orchestration of a large infrastructure of Remote Desktop Windows Servers**

Dissertation submitted in partial fulfillment  
of the requirements for the degree of

Master of Science in  
**Computer Science and Informatics Engineering**

Adviser: Paulo Orlando Reis Afonso Lopes, Assistant  
Professor, Faculdade de Ciências e Tecnologia  
da Universidade Nova de Lisboa

Co-adviser: Sebastian Bukowiec, IT Systems Engineer,  
European Organization for Nuclear Research (CERN)

Examination Committee

Chairperson: Prof. Doutor João Manual dos Santos Lourenço  
Rapporteur: Prof. Doutor José Henrique Pereira São Mamede  
Member: Prof. Doutor Paulo Orlando Reis Afonso Lopes



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

**December, 2017**



## **Orchestration of a large infrastructure of Remote Desktop Windows Servers**

Copyright © Ricardo Filipe Chaves Gaspar, Faculty of Sciences and Technology, NOVA University Lisbon.

The Faculty of Sciences and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.



*Dedico este trabalho à minha família que me apoiou ao longo desta jornada. Em especial, à minha mãe, Rosa, pelo seu amor e apoio incondicional mesmo à distância. À minha namorada, Ana Sofia, que embarcou comigo nesta aventura, me apoiou e mostrou o que é o amor. A todos os meus amigos que, mesmo longe, me acompanharam e apoiaram ao longo deste ano de aventura.*



## ACKNOWLEDGEMENTS

First, I would like to thank my faculty, Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa, and the Department of Informatics for the past few years of great teachings and opportunities. A special thanks to Professor Pedro Medeiros, coordinator of the Master degree in Computer Science, for supporting my application for the Technical Student Programme at CERN; and helping me with the needed bureaucracy so I could do my master thesis abroad.

Many thanks to my adviser, Professor Paulo Lopes, without whom it wouldn't be possible to write this thesis. Thanks for the help with the writing and the support along the way.

Secondly, I want to thank CERN for the opportunity I've been given; it was a long-time wish that became true. Of course, this was only possible thanks to my supervisor, Sebastian Bukowiec, who chose me to collaborate with him. I'm grateful for the things he taught me, the freedom and creativity to do my work and write my thesis, and for the great support and encouragement.

I would also like to thank my section leader, Michal Kwiatek, who challenged me to do a small Python project, one that I enjoyed and drove me to learn new things.

Thanks to my co-workers for the interesting conversations, the funny moments and the support.

Finally, thanks to my family and friends for the love and great support along this year.





## ABSTRACT

---

The CERN Windows Terminal Service infrastructure is an aggregation of multiple virtual servers running Remote Desktop Services, accessed by hundreds of users every day; it has two purposes: provide external access to the CERN network, and exercise access control to certain parts of the accelerator complex.

Currently, the deployment and configuration of these servers and services requires some interaction by system administrators, although scripts and tools developed at CERN do contribute to alleviate the problem. Scaling up and down the infrastructure (i.e., adding or removing servers) is also an issue, since it's done manually.

However, recent changes in the infrastructure and the adoption of new software tools that automate software deployment and configuration open new possibilities to improve and orchestrate the current service. Automation and Orchestration will not only reduce the time and effort necessary to deploy new instances, but also simplify operations like patching, analysis and rebuilding of compromised nodes and will provide better performance in response to load increase.

The goal of this CERN project, we're now a part of, is to automate provisioning (and decommissioning) and scaling (up and down) of the infrastructure. Given the scope and magnitude of problems that must be solved, no single solution is capable of addressing all; therefore, multiple technologies are required. For deployment and configuration of Windows Server systems we resort to Puppet, while for orchestration tasks, Microsoft Service Management Automation will be used.

**Keywords:** Windows Terminal Services, Remote Desktop, software configuration management, automation, orchestration, Puppet, Microsoft Service Management Automation.

---



## RESUMO

---

O CERN dispõe de uma infraestrutura que designa por “Windows Terminal Service Infrastructure”(WTS), e que é um agregado de servidores (que são, de facto, VMs) que executam os serviços de Remote Desktop e são acedidos diariamente por centenas de utilizadores. Os dois objectivos principais da WTS são: permitir o acesso aos utilizadores credenciados que se encontram *off-site*, e controlar o acesso ao software que manipula certas zonas do complexo onde se localiza o acelerador de partículas.

Neste momento a implantação (*deployment*) e configuração de serviços requer alguma interacção por parte dos administradores de sistema - embora com o auxílio de ferramentas e *scripts* desenvolvidos no CERN. As operações de aprovisionamento e remoção de servidores também requerem alguma interacção, o que contribui para piorar a situação atrás descrita.

Contudo, recentemente têm surgido novas ferramentas que permitem automatizar os processos de distribuição e configuração de software, bem como orquestrar múltiplas actividades desencadeadas sobre múltiplos alvos (por exemplo, servidores). Pode-se, assim, melhorar a qualidade dos serviços prestados pela infraestrutura, já que como resultado da introdução da automação e orquestração pode-se não só diminuir o tempo de aprovisionamento, mas também simplificar operações de *patching* e análise e reconstrução de nós problemáticos. Subsidiariamente, pode-se ainda conseguir reagir em tempo útil a aumentos de carga.

O objectivo deste projecto em que estamos inseridos é automatizar o aprovisionamento e remoção de servidores e serviços, e a escalabilidade da infraestrutura WTS. Considerando o âmbito e a magnitude dos problemas a resolver, não existe uma ferramenta única que, de uma assentada, os permita resolver todos, pelo que é necessário utilizar múltiplas tecnologias: para o aprovisionamento e configuração de sistemas Windows Server utilizaremos o Puppet; para orquestração de tarefas, vamos usar o Microsoft Service Management Automation.

**Palavras-chave:** Windows Terminal Services, Remote Desktop, gestão de configurações, automação, orquestração, Puppet, Microsoft Service Management Automation.

---

---

# CONTENTS

<b>List of Figures</b>	<b>xvii</b>
<b>List of Tables</b>	<b>xix</b>
<b>Glossary</b>	<b>xxiii</b>
<b>Acronyms</b>	<b>xxv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context . . . . .	1
1.2 Motivation . . . . .	3
1.3 Contributions . . . . .	4
1.4 Document Organisation . . . . .	5
<b>2 CERN's Windows Terminal Service Infrastructure</b>	<b>7</b>
2.1 The infrastructure . . . . .	7
2.1.1 CERN Cloud Infrastructure (OpenStack) . . . . .	8
2.1.2 Remote Desktop Services . . . . .	13
2.1.3 HAProxy . . . . .	17
2.2 Monitoring the WTS Operations: SCOM . . . . .	19
2.3 Management and Configuration of the WTS infrastructure . . . . .	20
2.3.1 Active Directory . . . . .	20
2.3.2 Group Policy . . . . .	21
2.3.3 Scripting . . . . .	22
2.3.4 Computer Management Framework . . . . .	23
<b>3 Agility</b>	<b>25</b>
3.1 DevOps . . . . .	25
3.1.1 Before DevOps . . . . .	26
3.1.2 What is DevOps? . . . . .	28
3.2 Infrastructure as Code . . . . .	33
3.2.1 Principles of IaC . . . . .	34
3.3 Configuration Management . . . . .	35

<b>4</b>	<b>Configuration Management</b>	<b>37</b>
4.1	SCM Tools: an Introduction . . . . .	37
4.2	SCM Tools: a brief survey . . . . .	39
4.2.1	Overview . . . . .	39
4.2.2	Puppet . . . . .	40
4.2.3	Ansible . . . . .	50
4.2.4	PowerShell DSC . . . . .	56
4.3	SCM Tools' Evaluation . . . . .	60
4.3.1	Tool Installation and Configuration . . . . .	60
4.3.2	Testing the Tools . . . . .	62
4.3.3	Integration with PowerShell DSC . . . . .	63
4.3.4	Evaluation Results . . . . .	66
4.4	WTS Configuration Management using Puppet . . . . .	68
4.4.1	<i>Puppet-wmi</i> module . . . . .	69
4.4.2	<i>puppet-sslcertificate</i> module . . . . .	69
4.4.3	<i>cernsslcertificate</i> module . . . . .	71
4.4.4	<i>teigi</i> module and <i>teigi_subfile</i> resource . . . . .	73
4.4.5	Building Puppet types from DSC modules . . . . .	75
4.4.6	WTS Puppet manifests . . . . .	77
<b>5</b>	<b>Automation and Orchestration</b>	<b>79</b>
5.1	Automation and Orchestration . . . . .	79
5.1.1	Runbooks and Workflows . . . . .	80
5.2	Evaluated Automation & Orchestration tools . . . . .	80
5.2.1	System Center Orchestrator . . . . .	81
5.2.2	Service Management Automation . . . . .	83
5.3	Windows PowerShell Workflows Concepts . . . . .	85
5.3.1	InlineScript . . . . .	87
5.3.2	Checkpoints . . . . .	89
5.3.3	Parallel Execution . . . . .	90
5.4	Service Management Automation usage at CERN . . . . .	91
5.5	Our Work with the SMA . . . . .	92
5.5.1	CERNOperations Integration Module . . . . .	92
5.5.2	Runbook <i>create-vm-with-volume</i> . . . . .	96
<b>6</b>	<b>Conclusions &amp; Future Work</b>	<b>101</b>
6.1	Conclusions . . . . .	101
6.2	Future Work . . . . .	103
	<b>Bibliography</b>	<b>105</b>
<b>I</b>	<b>Puppet test manifest</b>	<b>117</b>

II	Ansible test playbook	119
III	PowerShell DSC test configuration script	121
IV	DSC configuration script to set Group Policy rules	123
V	Manifest to set Group Policy rules using DSC resources	125
VI	Ansible playbook to set Group Policy rules using DSC resources	127
VII	Original version of <i>puppet-wmi</i> module	129
VIII	Improved version of <i>puppet-wmi</i> module	131
IX	Original version of <i>puppet-sslcertificate</i> module manifest	133
X	Original version of inspect.ps1.erb template	137
XI	Original version of import.ps1.erb template	139
XII	Improved version of <i>puppet-sslcertificate</i> module manifest	141
XIII	Improved version of inspect.ps1.erb template	145
XIV	Improved version of import.ps1.erb template	147
XV	<i>cernsslcertificate</i> module manifest	149
XVI	Windows provider for teigi_subfile	153
XVII	Improved teigi_subfile manifest to contemplate Windows systems	157
XVIII	Puppet Manifest to configure Remote Desktop Servers of CERN's WTS	161
XIX	Puppet Manifest to configure Remote Desktop License server	167
XX	Sync-Files PowerShell script to synchronise a folder structure	169
XXI	CERNOperations SMA Integration Module	171
XXII	<i>create-vm-with-volume</i> Runbook	189





## LIST OF FIGURES

1.1	Life cycle of a VM in the WTS infrastructure. . . . .	5
2.1	Traditional IT vs. Cloud computing service models [21]. . . . .	9
2.2	OpenStack components [22]. . . . .	11
2.3	OpenStack & WTS overview (adapted from [17]). . . . .	12
2.4	Remote Desktop Services Component Architecture. . . . .	14
2.5	CERN Terminal Service architecture. . . . .	15
2.6	CERN Windows Terminal Service connection workflow. . . . .	16
2.7	Remote Desktop Connection to CERN workstation. . . . .	16
2.8	Load balancing example showing the back-end concept [41]. . . . .	18
2.9	HAProxy+keepalived example. . . . .	19
2.10	Active Directory logical structure [51]. . . . .	21
2.11	Computer Management Framework logical structure [58]. . . . .	24
3.1	Waterfall model [62]. . . . .	26
3.2	Evolution of software development (adapted from [63]). . . . .	27
3.3	DevOps life cycle [68]. . . . .	31
4.1	Push and pull models [85]. . . . .	38
4.2	Puppet client-server model [90]. . . . .	41
4.3	An overview of a puppet run - only steps 2 to 5 (adapted from [97]). . . . .	48
4.4	Foreman Hostgroup example. . . . .	50
4.5	Ansible architecture overview [103]. . . . .	51
4.6	Ansible components [104]. . . . .	52
4.7	Inventory and playbooks in Ansible [106]. . . . .	54
4.8	PowerShell DSC push and pull models overview (adapted from [109]). . . . .	57
4.9	PowerShell DSC execution phase [111]. . . . .	57
4.10	PowerShell DSC push model [111]. . . . .	58
4.11	PowerShell DSC pull model [111]. . . . .	59
4.12	Group Policy GUI displaying the rule to set. . . . .	64
4.13	Group Policy GUI displaying the settings of the rule to set. . . . .	65
4.14	Cernsslcertificate logic flowchart. . . . .	74

4.15 Cernsslcertificate execution flowchart. . . . .	74
5.1 Orchestrator runbook example [133]. . . . .	80
5.2 Provisioning steps of a VM in the WTS infrastructure. . . . .	81
5.3 Orchestrator architecture [133]. . . . .	82
5.4 SMA architecture [14]. . . . .	84
5.5 SMA runbook execution steps [14]. . . . .	85
5.6 Execution of PowerShell commands on remote computers using InlineScript [141]. . . . .	88
5.7 Making a request to OpenStack through AIADM servers. . . . .	93
5.8 Flowchart of the <i>WaitForVM</i> function. . . . .	94
5.9 Flowchart of the <i>WaitForCMF</i> function. . . . .	95
5.10 Contact CMF Web Service to get the status of a VM's CMF Agent. . . . .	95
5.11 Invoking ai-kill to delete Puppet-managed servers on OpenStack. . . . .	95
5.12 Orchestrating the provisioning process of a VM in the WTS infrastructure. .	96
5.13 Flowchart of the <i>create-vm-with-volume</i> runbook. . . . .	99

## LIST OF TABLES

2.1	Subset of CERN's OpenStack flavours. . . . .	12
3.1	Conceptual framework characterising DevOps (adapted from [4]). . . . .	28
3.2	How DevOps addresses different challenges (adapted from [64]). . . . .	29
3.3	DevOps practices (adapted from [2]). . . . .	30
4.1	Foreman Hostgroups. . . . .	49
4.2	Puppet code and hostgroup organisation. . . . .	50
4.3	Characteristics of the evaluated SCM tools. . . . .	61
4.4	Evaluation results for the surveyed SCM tools. . . . .	68



## LIST OF LISTINGS

1	Puppet (hostgroup) manifest example for Windows systems. . . . .	44
2	Another Puppet (hostgroup) manifest example for Windows systems. . .	46
3	YAML syntax examples [105]. . . . .	53
4	Example of an inventory file using the INI format. . . . .	54
5	Example of an inventory file using the YAML format. . . . .	55
6	Playbook with one play. . . . .	55
7	Playbook with two plays. . . . .	55
8	PowerShell DSC configuration script example [112]. . . . .	60
9	PowerShell DSC configuration script to set a Group Policy rule. . . . .	65
10	Puppet manifest using a DSC to set a Group Policy rule. . . . .	66
11	Ansible playbook using a DSC to set a Group Policy rule. . . . .	67
12	Original version of <i>puppet-wmi</i> module. . . . .	70
13	Improved version of <i>puppet-wmi</i> module [119]. . . . .	71
14	Improved version of <i>puppet-sslcertificate</i> module manifest. . . . .	72
15	Sample of <i>cernsslcertificate</i> module showing the use of resource collectors.	74
16	Docker file to create a container to build Puppet types from DSC mod- ules [128]. . . . .	76
17	Sample manifest using DSC resources . . . . .	77
18	Windows PowerShell Workflow basic structure (adapted from [141]). . .	86
19	Windows PowerShell Workflow calling a function. . . . .	87
20	InlineScript syntax [141]. . . . .	87
21	Formatting a new volume in a remote computer using an InlineScript [141].	88
22	Passing and returning values from an InlineScript activity. . . . .	89
23	Checkpoint-Workflow example. . . . .	89
24	An example using the Parallel block. . . . .	90
25	An example using the Parallel block with a sequence of commands. . . .	91
26	An example using the ForEach -Parallel loop. . . . .	91



## GLOSSARY

**Ceph** Ceph is a software storage platform that implements object storage on a distributed computer cluster and provides interfaces for object-, block- and file-level storage. Ceph aims primarily for completely distributed operation without a single point of failure, scalable to the exabyte level, and freely available. It replicates data and makes it fault-tolerant, using commodity hardware and requiring no specific hardware support. As a result of its design, the system is both self-healing and self-managing, aiming to minimise administration time and other cost [32].

**Docker** Docker is a software technology providing containers, promoted by the company Docker, Inc. It provides an additional layer of abstraction and automation of operating-system-level virtualisation on Windows and Linux. Docker uses the resource isolation features of the Linux kernel such as *cgroups* and kernel *namespaces*, and a union-capable file system (e.g. OverlayFS) to allow independent “containers” to run within a single Linux/Windows instance, avoiding the overhead of starting and maintaining virtual machines (VMs) [79].

**MOF** The Managed Object Format (MOF) defined by DMTF is a file format with its own language based on IDL (the Object Management Group’s Interface Definition Language). It provides a way to describe object-oriented class and instance definitions in textual form, with the goals of human readability and parsing by a compiler. The main components of a MOF specification are textual descriptions of element qualifiers (meta-data about classes, properties, methods, etc.), comments and compiler directives, and the specific class and instance definitions [108].

**SOAP** Simple Object Access Protocol (SOAP) is a lightweight protocol for exchange of information in a decentralised, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses [101].

**Windows Workflow Foundation** Windows Workflow Foundation (WWF) is a programming model, set of tools, and runtime environment that allows to write declarative workflows on the Windows platform to represent the execution model of programs.

It provides an API, an in-process workflow engine, and a designer to implement long-running processes as workflows within .NET applications [143].

**WinRM** Windows Remote Management (WinRM) is the Microsoft implementation of WS-Management Protocol, a standard firewall-friendly protocol that allows hardware and operating systems, from different vendors, to interoperate [99].

**WMI** Windows Management Instrumentation (WMI) is the Microsoft implementation of Web-Based Enterprise Management (WBEM), which is an industry initiative to develop a standard technology for accessing management information in an enterprise environment. WMI uses the Common Information Model (CIM) industry standard to represent systems, applications, networks, devices, and other managed components. CIM is developed and maintained by the Distributed Management Task Force [55].

**WS-Man** Web Services Management (WS-Man) provides interoperability between management applications and managed resources, and identifies a core set of web service specifications and usage requirements that expose a common set of operations central to all systems management. It is a SOAP-based protocol for managing computer systems (e.g. personal computers, workstations, servers, smart devices) that supports web services and helps constellations of computer systems and network-based services collaborate seamlessly [100].



## ACRONYMS

**ACL** Access Control List.

**AD** Active Directory.

**API** Application Programming Interface.

**CD** Continuous Delivery.

**CDev** Continuous Development.

**CERN** Conseil Européen pour la Recherche Nucléaire.

**CI** Continuous Integration.

**CLI** Command-line Interface.

**CM** Continuous Monitoring.

**CMF** Computer Management Framework.

**CPU** Central Processing Units.

**CT** Continuous Testing.

**DevOps** Development and Operations.

**DMTF** Distributed Management Task Force.

**DSC** Desired State Configuration.

**DSL** Domain Specific Language.

**FA** Feedback Agent.

**GPO** Group Policy Object.

**GUI** Graphical User Interface.

**GUID** Global Unique Identifier.

**HEP** High Energy Physics.

**IaaS** Infrastructure-as-a-Service.

**IaC** Infrastructure-as-Code.

**IP** Internet Protocol.

**IT** Information Technologies.

**LCM** Local Configuration Manager.

**NIST** National Institute of Standards and Technology.

**NSS** Name System Set.

**OS** Operating System.

**OU** Organisation Unit.

**QA** Quality Assurance.

**RDCB** Remote Desktop Connection Broker.

**RDG** Remote Desktop Gateway.

**RDL** Remote Desktop Licensing.

**RDP** Remote Desktop Protocol.

**RDS** Remote Desktop Services.

**RDSH** Remote Desktop Session Host.

**RDVH** Remote Desktop Virtualization Host.

**RDWA** Remote Desktop Web Access.

**SaaS** Software-as-a-Service.

**SCM** Software Configuration Management.

**SCOM** System Center Operations Manager.

**SDN** Software Defined Network.

**SMA** Service Management Automation.

**SRS** Software Requirements Specification.

**SSL** Secure Sockets Layer.

**TDD** Test-Driven Development.

**URI** Unique Resource Identifier.

**VCS** Version Control System.

**VM** Virtual Machine.

**WMF** Windows Management Framework.

**WTS** Windows Terminal Service.



## INTRODUCTION

### 1.1 Context

As one can imagine, an organisation like Conseil Européen pour la Recherche Nucléaire (CERN), whose main goal is to do research in the field of High Energy Physics (HEP) and has more than 16000 employees from 22 countries, requires a huge amount of computing resources to carry out its research and support daily operations - two Data Centres (one in Geneva, Switzerland and another in Budapest, Hungary) with roughly 12,500 physical servers and 25,000 virtualised servers .

This document focuses on CERN's Windows Terminal Service (WTS), a subset of these computing resources that provides user access to the CERN intranet and controls the access to certain parts of the accelerator complex. The CERN's WTS is, furthermore, virtualised: its servers are virtual machines (VMs) organised in multiple groups (referred to as clusters) and assigned to different user groups and departments.

Installing, configuring and managing a large number of machines (virtual or not), some of them with quite different configuration parameters, requires a lot of effort from systems administrators (SysAdmins) mainly because there's a significant amount of human interaction. For example, to deploy a new cluster for, e.g., a CERN department, multiple servers must be deployed, networking must be set up, and several Windows software components and/or services must be configured. The current approach to minimise the SysAdmin's work is to use multiple services and tools:

- directory services (Active Directory) - to manage users, groups, computers and other devices;
- group policies (Group Policy) - to enforce policies upon users, groups and computers;

- scripts - to automate some deployment and configurations tasks;
- configuration management software - to install and/or remove software applications.

CERN's WTS current management approach relies on the previous mentioned tools plus scripts and an in-house developed configuration management tool, dubbed Computer Management Framework (CMF) (see section 2.3.4) to automate several types of actions; however, there is still a lot of room for improvement.

Unfortunately Windows Server components have distinct interfaces (command line, Graphical User Interface (GUI), Application Programming Interface (API), etc.) which make the scripting approach a possible but tedious solution, as each resulting script deals with a particular component and has little opportunity for reuse. So, one must take a step back and reexamine the problem of Configuration Management.

We begin by stating the obvious: what system administrators want is to have a compliant (all the machines in a group have the "same" target software configuration), scalable and easy to manage infrastructure. Therefore, the main objective is the orchestration of common activities by defining workflows to reduce error-prone manual actions and operational workload.

To address this goal, given the size and growth rate of IT infrastructures all over the world, a new paradigm of configuration management was born: Infrastructure-as-Code (IaC) [1], itself a descendant of the Development and Operations (DevOps) paradigm [2–6]. IaC states that configuration management problems can be solved using the same concepts of software development: the idea is to write programs that specify the desired configuration states, and then execute them.

The result is the availability of a number of Software Configuration Management (SCM) [7] tools to tackle these challenges, like Puppet [8], and Chef [9]. The first incarnation of those tools was targeted at server architectures running UNIX-like operating systems, since they represented the majority when compared to, e.g., Windows. This has been reflected in the way they operate and manage resources since, in UNIX-like systems, most configurations are stored in files and software is installed via packages. Only in recent years these tools have begun to support Windows, and that process is ongoing, with an increased number of features available in every new version of a tool. Windows server deployments (core versions excluded) were mostly based on GUI and their configuration and management has traditionally required a lot of user interaction. However, in recent versions of Windows Server, many efforts have been made by Microsoft to offer tools like PowerShell extensions (cmdlets, in their terminology) such as Desired State Configuration (DSC) [10] that allow programmers to create scripts that require no user interaction to deploy software packages and configure system components.

Another important aspect is orchestration, the ability to automate repetitive tasks and complex processes in a way that they can be carried out by computers, i.e. without human interaction.

Note that automation, as carried out by configuration management software tools, is concerned with a single task (e.g. installing an application, configuring a web server or stopping a service) while orchestration is concerned with the execution of a process or workflow - multiple tasks that can involve multiple systems. An example of a workflow can be setting up an infrastructure for a complex service consisting of several, simpler ones: a web server, an application server and a database server [11].

The usage of automation and orchestration tools may significantly improve the management of large computing infrastructures, as well as reduce the systems administrators' workload. Some use cases can be alert remediation - monitor a resource and react to a trigger -, maintenance tasks and cross-technology integration<sup>1</sup> [12]. For Windows Server systems, Microsoft has been releasing products that address those needs, like Orchestrator [13] and, more recently, Service Management Automation (SMA) [14].

## 1.2 Motivation

Within CERN's Information Technologies (IT) Department, the Applications and Devices group is the one responsible for the Windows Terminal Service infrastructure which, as referred previously, grants access to servers in the CERN intranet to authorised users. These servers are grouped in clusters which can be categorised in two types: General Purpose (a.k.a. Public) clusters and Specialised (a.k.a. Dedicated) clusters.

There are 3 General Purpose clusters with a total 21 servers which goal is to allow CERN users to remotely log in and use the pre-installed productivity software (Microsoft Office, etc.) and provide access to resources available only in the CERN intranet . The Specialised clusters, for which there are there are 30 (with a total of circa 130 servers), are dedicated to different teams and departments as they serve different purposes related to specific tasks such as accelerator's experiments, beam monitoring, etc.. A more detailed explanation is described in section 2.1.

The Applications and Devices team must deploy, configure and manage these servers and, currently, a sizeable amount of that work requires human interaction; and, to worsen things up, these tasks must be performed quite often. The challenge, therefore, resides in how to maintain compliance and adherence to CERN's standards and best practices in a vast, highly dynamic, and growing infrastructure.

In 2011, CERN's IT Department began to review the data centre infrastructure in order to expand and optimise the its usage [15]. The goal was to explore widely-used technologies for cloud computing, virtualisation, and configuration management to reduce the operational effort, improve agility and support lights-out (i.e., no human presence) remote data centres. Hence, the birth of the CERN's Agile Infrastructure project [15, 16] which aimed to deploy in CERN data centres, with minimal customisation, the same set of tools and processes for data centre management that was used in data centres elsewhere.

---

<sup>1</sup>Integration with other software technologies or systems.

At that time, OpenStack [17, 18] - a software stack that can be used to provide Infrastructure-as-a-Service (IaaS) private clouds- was chosen for CERN data centres. As the IT Department embraced that new paradigm, it was decided that the WTS infrastructure should also be deployed on top of OpenStack.

The reasons for adopting a private cloud infrastructure are manifold, but we will point just two: the sharing of hardware resources (compute, storage and networking) among cloud consumers; and its intrinsic ability to rapidly deploy (and retire) a virtual machine - either through cloning of a base image (a.k.a. template or golden image), or from scratch. In short, WTS servers are Windows Server VMs that can be rapidly deployed with OpenStack; but, after the initial deployment step, further configuration and/or installation of software components must be performed. To automate these tasks, ensure compliance across servers providing the same services and reduce administrator burden, Puppet was chosen.

Since 2013 the Agile Infrastructure is in place and continues to evolve [16] but now, with the introduction of Puppet versions with support for the Windows platform together with Microsoft changes that enable its server tools to be used in non-interactive ways, an opportunity arises to extend those benefits to the Windows Terminal Service infrastructure.

Therefore, our goal is to leverage automation and orchestration across WTS to reduce error-prone manual actions and operational workload. By endowing the infrastructure with automated mechanisms it is possible to reduce both the time to provision new servers and the number of misconfiguration events.

Fig. 1.1 depicts the life cycle of a VM in the WTS showing the steps to provision and maintain it as well as the tools required for these steps. In the next section we will further detail our contributions to the WTS infrastructure including the process illustrated in the fig. 1.1 using the tools previously mentioned.

### 1.3 Contributions

Our aim is to address the challenges of configuration management, automation and orchestration using an IaC approach in order achieve an automated infrastructure with less operational costs. To reach the above goals, and as there is no single “silver bullet” solution that solves them all, several technologies had to be used together: Puppet for configuration management and SMA for automation and orchestration.

With the adoption of these technologies a reduction has been achieved in the number of operations requiring human (system administrator) intervention for:

- WTS components (servers, software, services) provisioning and decommissioning;
- remedial services (taking a component out and refreshing it or reinstalling it);
- scaling up (and down) the infrastructure as a response to user load changes.



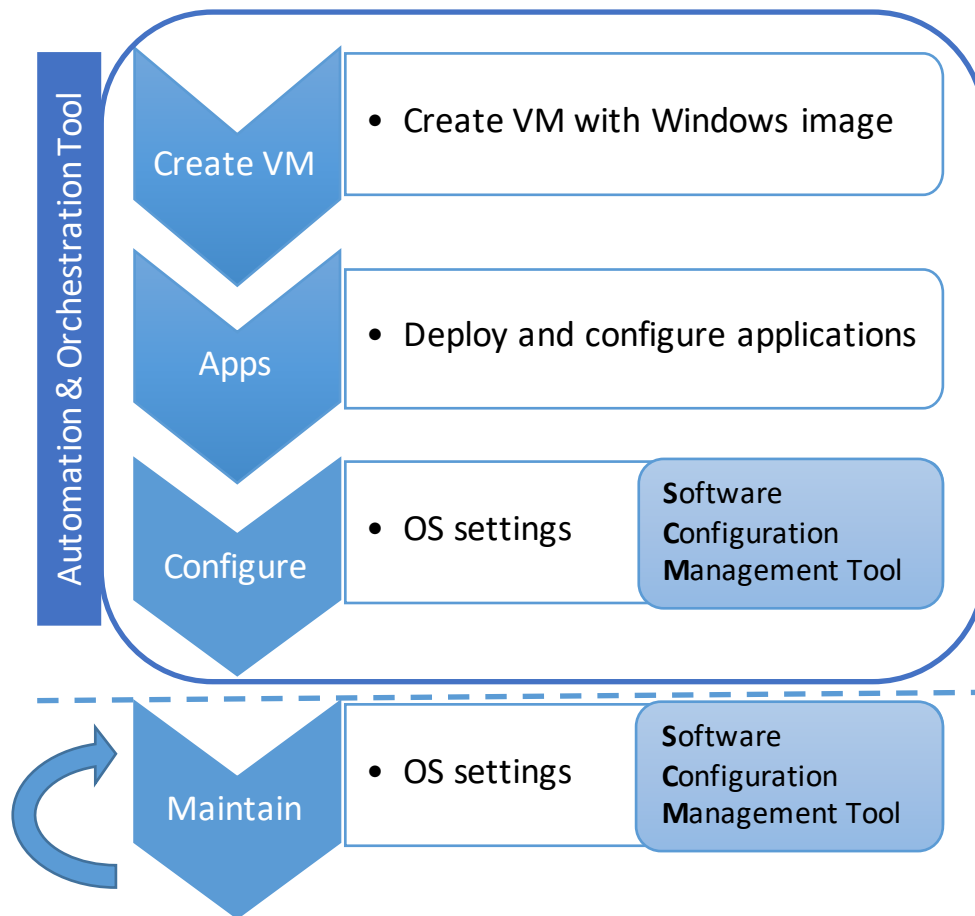


Figure 1.1: Life cycle of a VM in the WTS infrastructure.

As a result of the use of Puppet for configuration management of the WTS VMs (Windows servers), a number of improvements were made to some Puppet community modules targeted to Windows systems, namely *puppet-sslcertificate* and *puppet-wmi*, and also to modules privately used at CERN, such as *cernsslcertificate* and *teigi*.

Other contributions were made as a result of using Service Management Automation - the automation and orchestration tool. A PowerShell module named *CERNOperations* was developed to abstract the most common operations invoked by other SMA runbooks (e.g. Delete VM, Delete Volume, Wait for VM, Wait for CMF, etc.). Another major contribution was a runbook to orchestrate the provisioning of a WTS VM, following the steps depicted on fig. 1.1.

## 1.4 Document Organisation

The remainder of this document is organised as follows:

Chapter 2 describes the infrastructure of CERN's Windows Terminal Service and the software technologies currently used to provide those services. It explains the WTS architecture, its purpose, the role of each component, and how the infrastructure is

managed.

Chapter 3 starts by describing the birth of DevOps and explains the concepts associated with it, as configuration management technologies play an important part in this paradigm, one that relies on automation and collaboration between development and operations teams to deliver properly configured infrastructures to run applications and services. It also introduces the motivations for the existence of tools that automate the configuration of large infrastructures.

Chapter 4 starts with a brief introduction of SCM tools, and then presents and describes some of the tools available to increase the level of automation in WTS and makes a comparison between them. It concludes by presenting the work done using Puppet, the tool that was selected to configure and manage the WTS infrastructure.

Then, Chapter 5 begins with the introduction of a set of tools that were selected as candidates for automation and orchestration of the WTS infrastructure, followed by a more detailed presentation of the SMA and how it was used to develop workflows to automate the provisioning of servers and execution of maintenance tasks.

Finally, Chapter 6 discusses not only the WTS infrastructure needs and the solutions that were put in place to address them, but presents other possible solutions that could fit the same purposes. It finishes with a section dedicated to the discussion possible improvements and technologies that should be explored in a near future to increase automation to the WTS infrastructure even more.

## CERN'S WINDOWS TERMINAL SERVICE INFRASTRUCTURE

CERN's Windows Terminal Service (WTS) infrastructure has two major purposes: provide access to the CERN network from the outside and control the access to certain parts of the accelerator complex. This chapter presents the WTS architecture and the software technologies that CERN's IT Department is currently using to deploy, configure and manage WTS hosts and services.

### 2.1 The infrastructure

The WTS is an infrastructure of virtualised hosts that, for the sake of simplicity of this document, are grouped into two categories that we shall refer to as *infrastructure services* (e.g., proxy, broker and licensing hosts) and *user services* (i.e., hosts that are accessed by users to run applications, etc.). Hosts running user services are grouped in clusters and assigned to different user groups and departments at CERN; they all run the Windows Server operating system and allow incoming remote connections in order to provide the same user experience as if they were accessed locally. Remote access to a Windows Server host can be enabled through the activation of a server role - a software service embedded in the operating system - called Remote Desktop Services [19].

With RDS, users can access resources in CERN's internal network using a secure connection without providing direct access to these resources, i.e., the user's computer (accessing the remote server's desktop) has no access to the internal resources but the user, indirectly through the remote server, can access those resources. A more detailed description can be found in section 2.1.2.

In WTS there are two types of user-accessible clusters: public and dedicated.

Public clusters (for which there are three, with a total of 21 servers) allow any user with a valid CERN account to log in and use one of the available hosts; these hosts are pre-installed with productivity software (Microsoft Office, etc.) and enable users' access to resources available only in the CERN intranet (e.g. documents in file servers, internal websites).

Dedicated clusters (for which there are 30, with a total of circa 130 servers) have different purposes, since they are assigned to different groups at CERN and their access is restricted (e.g. monitoring services for the experiments). Each cluster (i.e., its member servers) requires its own software stack since they have different purposes. Only users that are members of a group can access the group's dedicated cluster(s).

In public clusters, servers accessed by the users are pre-installed and ready-to-use, while servers in dedicated clusters are handed out to group/team managers and assigned administrative privileges that let them deploy new software. Nevertheless, the hosts in the dedicated clusters also have to follow general rules and policies that are in place to ensure a compliant and secure infrastructure.

Gateway servers are also a part of CERN's WTS; they run a service that enables users to remotely connect directly to their (physical) workstations. In this case, users can remotely work on their workstations located at CERN in the same way they would do if they were in their office.

Given the size of this infrastructure, it makes sense to have tools that automate the deployment and configuration, and provide mechanisms that (re-)apply configuration parameters (either because they have been inadvertently or maliciously altered or because a new configuration must be pushed).

The remaining subsections describe the technologies that are in place to support the WTS infrastructure, and how they are used.

### 2.1.1 CERN Cloud Infrastructure (OpenStack)

As previously mentioned, our WTS is run as a virtualised infrastructure where the VMs are executed on top an IaaS private cloud; therefore, we would like to start introducing two concepts - cloud computing and IaaS - quoting the National Institute of Standards and Technology (NIST) definitions (the bold markup being ours):

“Cloud computing is a model for enabling ubiquitous, convenient, **on-demand** network access to a **shared pool** of **configurable** computing resources (e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned and released** with **minimal management effort or service provider interaction**.” ([20])

Furthermore, in a cloud environment we have several important entities (and/or roles) but, for brevity, we refer only three: the (service) provider, the (service) consumer and the user. They interact with “the cloud” in different ways: the user, simply uses the applications, and sometimes has no idea or does not care if they run on a cloud, or not; the

consumer usually interacts with the cloud software through portals and requests services (e.g., in a IaaS - see below - service model, the consumer asks for a Virtual Machine (VM) with some characteristics); the provider manages the cloud resources (physical infrastructure, software, etc.) and makes them available to consumers.

Among the different types of service models provided by clouds (see fig. 2.1), the most basic is the Infrastructure as a Service (IaaS). In IaaS, “the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where **the consumer is able to deploy and run arbitrary software**, which can include operating systems and applications. The consumer **does not manage or control the underlying cloud infrastructure** but has control over operating systems, storage, and deployed applications.” ([20])

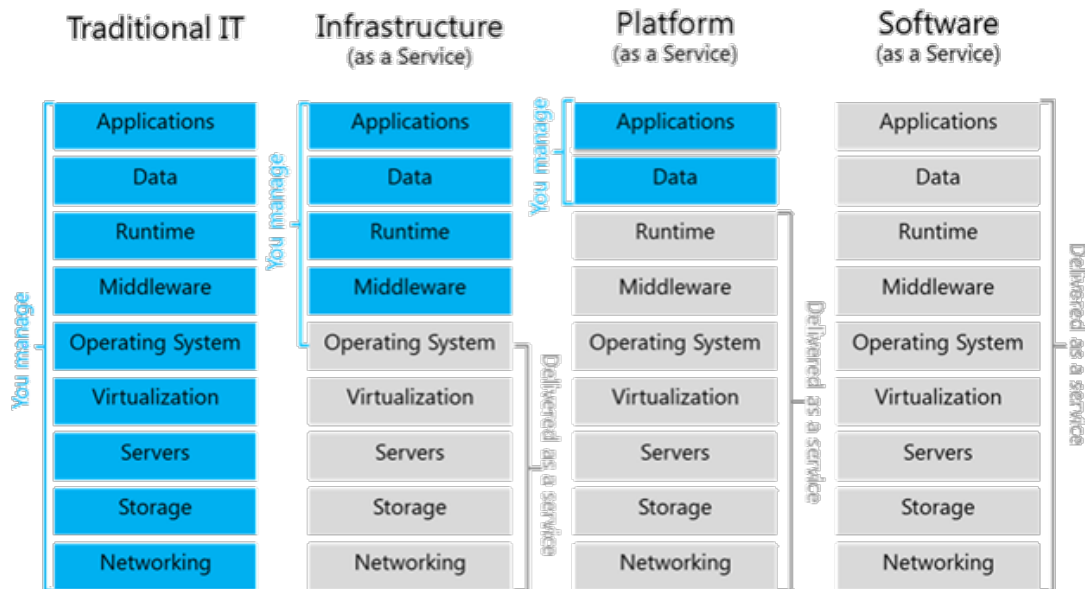


Figure 2.1: Traditional IT vs. Cloud computing service models [21].

The main reasons for using a virtualised, cloud-based infrastructure relates to the flexibility and efficiency it provides: existing hardware may be more efficiently used since it is possible to have multiple VMs per physical host with just a small increase in power usage; and resource sharing is what gives the flexibility for this approach, since it provides the means to allocate resources to those who need them (e.g., new VMs are created) and deallocate/recall them from those which need less, or do not need them anymore (e.g., some VMs are destroyed).

CERN uses OpenStack [18, 22] as its cloud software stack for the two data centres (DC) it owns: one in CERN’s headquarters in Geneva, Switzerland, and another at the Wigner Research Centre for Physics in Budapest, Hungary. The last one was built as an expansion of Geneva’s DC to increase CERN computing capabilities. They are about 1000 Km apart and connected through 100Gb/s (Gigabit per second) [23] optical fibres. The

VMs for CERN's Windows Terminal Service are hosted in the Meyrin site (Geneva DC).

OpenStack is composed by a set of independent community projects which provide different services and are coupled together to assemble a private cloud infrastructure. Their independence comes from the fact that they can be updated independently, thus having different versions, and not all are required to build a cloud infrastructure. These projects can be also designated as components or services interchangeably. OpenStack features the following main components/projects (adapted from [22]):

- **Compute (a.k.a. Nova)** allows the user to create and manage a large number of VMs using a set of predefined images. It is the “brain” of the cloud. Nova facilitates this management through an abstraction layer that interfaces with supported hypervisors.
- **Block Storage (a.k.a. Cinder)** provides persistent block storage for compute instances (VMs). Cinder creates software-defined storage via abstraction by virtualising pools of block storage from a variety of back-end storage devices which can be either software implementations (e.g. Ceph) or traditional hardware storage products. Cinder's primary functions are to manage the creation, attaching and detaching of the block devices. The consumer requires no knowledge of the type of back-end storage equipment or where it is located [24]. This flexible architecture makes creating and managing block storage devices very easy.
- **Networking (a.k.a. Neutron)** provides various networking services to cloud users (tenants) such as IP address management, DNS, DHCP, load balancing, and security groups (network access rules, like firewall policies). It also provides a framework for Software Defined Networks (SDNs) that allows for pluggable integration with various networking solutions. Neutron allows cloud tenants to manage their guest network configurations. Security features include network traffic isolation, availability, integrity and confidentiality.
- **Image Service (a.k.a. Glance)** provides disk image management services: image discovery, registration, and delivery to the Compute service, as needed. Users can upload and discover data assets that are meant to be used with other services. This currently includes VM images and metadata definitions. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image.
- **Object Storage (a.k.a. Swift)** provides support for storing and retrieving arbitrary data in the cloud. It is important to understand that object storage differs from traditional file system storage. It is best used for static data such as media files (MP3s, images, videos), virtual machine images, and backup files. These blobs of data (objects) are stored in an organisational hierarchy that offers anonymous read-only access, Access Control List (ACL) defined access, or even temporary access. Applications can store and retrieve data via an HTTP RESTful API. In addition, it

offers an Amazon Web Services S3 compatible API. Swift provides a high degree of resiliency through data replication and can handle petabytes of data.

- **Identity Service (a.k.a. Keystone)** provides a central directory of users mapped to the OpenStack services. It is used to provide an authentication and authorisation service for other OpenStack services.
- **Dashboard (a.k.a. Horizon)** is a web-based portal that interacts with all the underlying OpenStack services such as Nova, Neutron, etc. Through this interface cloud administrators and tenants (users) can provision, manage, and monitor cloud resources.

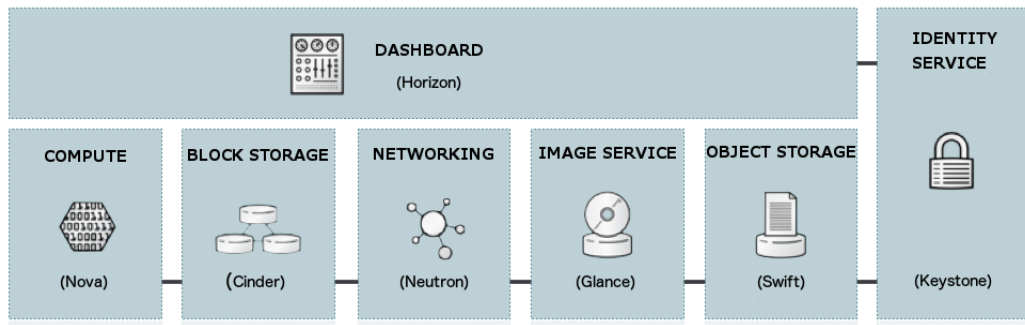


Figure 2.2: OpenStack components [22].

CERN’s OpenStack infrastructure has all components deployed, with the exception of Swift and a few others not referred here (e.g. Ceilometer and Heat) [18].

As previously described, OpenStack Nova supports multiple hypervisors [25], such as KVM [26], Xen [27], Hyper-V [28] and VMWare ESXi [29]. For the sake of comprehension, it’s important to introduce here the concept of hypervisor: an hypervisor is a software module that abstracts (virtualises) hardware resources and multiplexes and manages them so that multiple Operating Systems (OSs) can run simultaneously on the same physical computer. Bugnion et al. [30] define an hypervisor as a specialised piece of system software that manages and runs virtual machines; the concept was first introduced by Popek and Goldberg [31].

Nowadays, WTS VMs are run on top of the KVM hypervisor as KVM’s current versions also support Windows guests; previously, both KVM (for Linux guests) and Hyper-V (for Windows guests) were used, but after some time CERN’s OpenStack team chose to adopt one hypervisor only, because that simplifies the development and management.

CERN’s private cloud provides different flavours to host VMs. A flavour defines the VM “hardware” characteristics (e.g., of number of virtual Central Processing Units (CPU) cores, amount of memory and disk size). Currently, there are 5 flavours available for CERN’s WTS (see table 2.1).

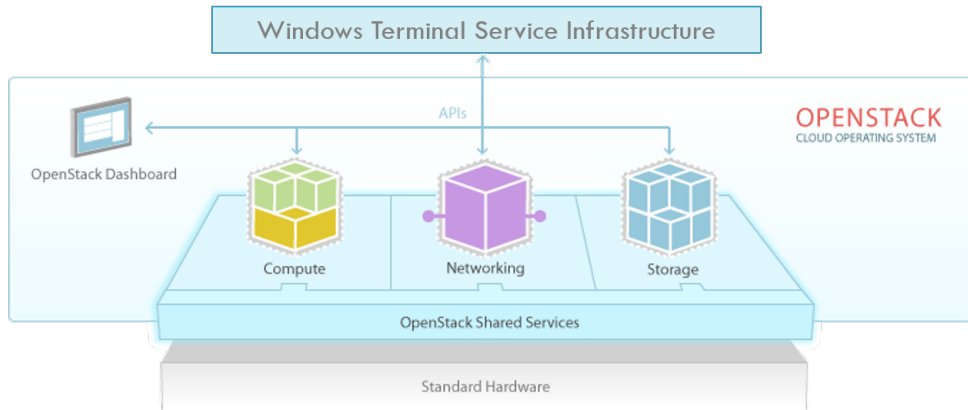


Figure 2.3: OpenStack &amp; WTS overview (adapted from [17]).

Table 2.1: Subset of CERN's OpenStack flavours.

Name	Memory (MB)	Disk (GB)	vCPU's (Cores)
m2.small	1875	10	1
m2.medium	3750	20	2
m2.large	7500	40	4
m2.xlarge	15000	80	8
m2.2xlarge	30000	160	16

In WTS, all Windows VMs use the m2.xlarge flavour since it provides enough disk space to store the Windows operating system, and also sufficient CPU and memory resources to execute it. The use of more generous flavours is possible, but due to the amount of resources they require, it's often difficult to find available resources to allocate. In addition, the large number of VMs already used for WTS is continuously growing and the team responsible for WTS infrastructure has adopted a horizontal scaling (having more machines) approach as opposed to vertical scaling (have less machines but with more resources) to ensure a good availability and quality of service.

OpenStack images (a.k.a golden images or templates) are managed through Glance, but stored on a Ceph [32] back-end. When an image is chosen to start a VM, the instance's (virtual) system disk is created locally on the host that is going to execute the VM; the host's local storage for those disks is based on SSD technology. However, if the instance needs more storage, that space is assigned as a new disk volume, with a custom size; usually additional volumes are crafted from Ceph's HDD-based storage pools.

CERN uses Ceph as the back-end storage service due to its superior design with respect to reliability, scalability (in terms of size) and future growth (in terms of use cases) [33]. CERN has chosen to integrate Ceph in the OpenStack architecture through Cinder, the OpenStack service responsible for managing block storage and providing volumes to VMs.



**Note:** The purpose of this document is not to describe the OpenStack-based CERN's private cloud in detail, but rather describe how the cloud is used to host the different types of (virtual) servers that are used to build the WTS infrastructure. For a more detailed description of OpenStack, see [22]. To learn more about CERN's OpenStack private cloud, see [18, 33–36].

### 2.1.2 Remote Desktop Services

Microsoft's Remote Desktop Services (RDS) [19] are implemented in a role included in Windows Server; when enabled, it allows remote connections to access and control Windows machines over the network. Previous to Windows Server 2008 R2 the role was named Terminal Services; CERN's IT Department has kept the original name, hence the WTS acronym: "CERN Windows Terminal Service".

**Note:** Throughout this document the service provided by the IT Department at CERN will be referred to as "CERN's Windows Terminal Service" but, for the sake of convenience, "Remote Desktop Services Infrastructure" or, in short, "RDS Infrastructure" will also be used to refer to an infrastructure running these services.

RDS uses the Remote Desktop Protocol (RDP) [37, 38] to establish connections between clients and servers, enabling users to connect to remote computers and interact with them as if they were "right there", at the user's desk where the computer is located. Users can connect to remote computers using thin-client applications such as Remote Desktop Connection or RemoteApp.

In RDP, a client receives the (screen) image of the remote system's desktop, displays that image locally, and sends the input from the local mouse and keyboard back to the remote system, which receives that input and responds accordingly, as if the input was coming from directly connected devices. As a result, all execution takes place on the remote system.

Figure 2.4 depicts the six fundamental components of the Remote Desktop Services architecture (the Server Manager is not a part of it, it's just a management console included in Windows Server systems).

- **Remote Desktop Session Host (RDSH):** formerly named Terminal Server, enables a server to host remote desktop sessions. Users can connect to an RDSH server to execute programs, manage files and use network resources on that server. The access can be performed through the Remote Desktop Connection client.
- **Remote Desktop Connection Broker (RDCB):** previously named Terminal Server Session Broker, keeps track of user sessions in a load-balanced RDSH server farm. It uses a database to store session state information that includes session IDs, their associated usernames, and the name of the server where each session resides. When, in a load-balanced farm, a user currently connected to an RDSH server requests a

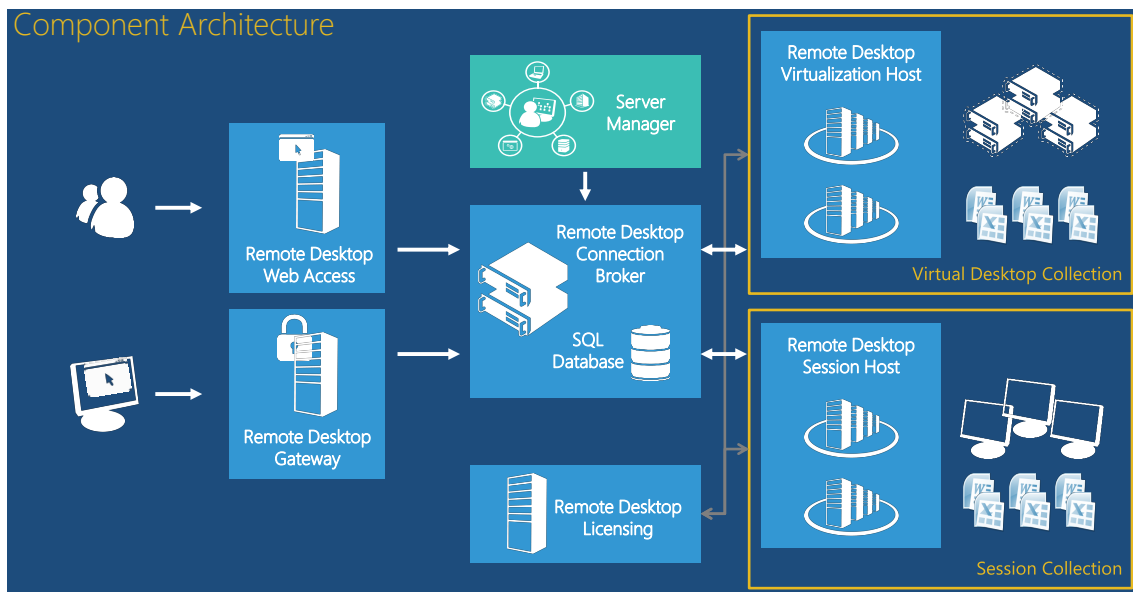


Figure 2.4: Remote Desktop Services Component Architecture.

new connection, the Connection Broker redirects the user to the same Session Host server; this prevents the user from being connected to a different server in the farm, thus starting a new session.

- **Remote Desktop Web Access (RDWA):** once named Terminal Server Web Access, allows users to make Remote Desktop connections to remote servers using a web browser.
- **Remote Desktop Gateway (RDG):** formerly named Terminal Server Gateway, enables authorised users to connect to resources on an internal corporate network over the Internet or on the same network. The network resources can be RDSH servers running virtual desktops, or physical computers with Remote Desktop Services enabled.
- **Remote Desktop Licensing (RDL):** named Terminal Server Licensing in the past, manages the Remote Desktop Services client access licenses (RDS CALs) that are required for each device or user to connect to an RDSH server. It is used to install, issue, and track the availability of CALs on a Remote Desktop license server. It only allows remote connections if there are licenses available, otherwise they are rejected.
- **Remote Desktop Virtualization Host (RDVH):** integrated with Microsoft Hyper-V to host VMs and provide them to users as virtual desktops. It can be configured so that each user is assigned a unique virtual desktop, or they are dynamically assigned to a shared pool of virtual desktops.

The above is just a general description of RDS; CERN's Windows Terminal Service infrastructure has a somewhat different structure, illustrated in fig. 2.5 for two use cases: the first scenario (the upper-half of the figure) shows how remote desktop connections to public/dedicated servers are dealt with; the second portrays remote desktop access to physical Windows 10 workstations, placed in user's desks.

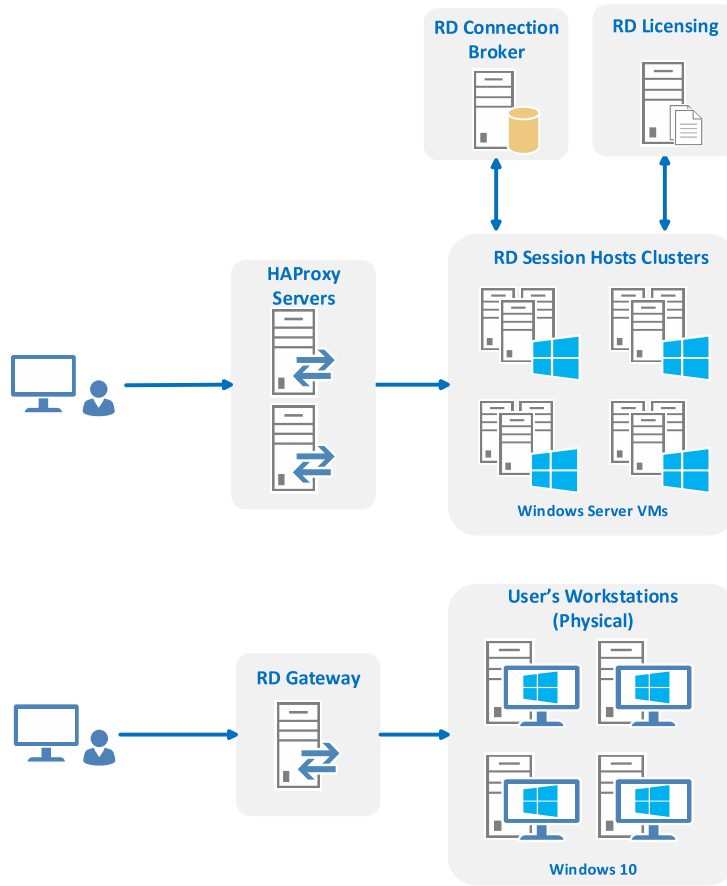


Figure 2.5: CERN Terminal Service architecture.

In the first scenario (see fig. 2.6), a cluster of HAProxy [39, 40] servers running on top of Linux VMs, acts both as a proxy and a load balancer for the RDP traffic, effectively replacing RDCB which is used in CERN's WTS just to keep track of user sessions. When a remote desktop connection, initiated at the user's client (1) reaches one HAProxy server, the load balancer chooses one of the RDSH servers (2) to try to establish the remote desktop connection. To do that, the chosen RDSH server has to request a license from RDL (3) and then ask if the RDCB has any connection state for that user (4); if one is found, the session is resumed; if not, a new session is created. Afterwards (5), the remote desktop connection is established.

In the second scenario (see fig. 2.7), RDG is used to connect to the workstations (user desktops), while checking for license availability. Here, a remote desktop connection is

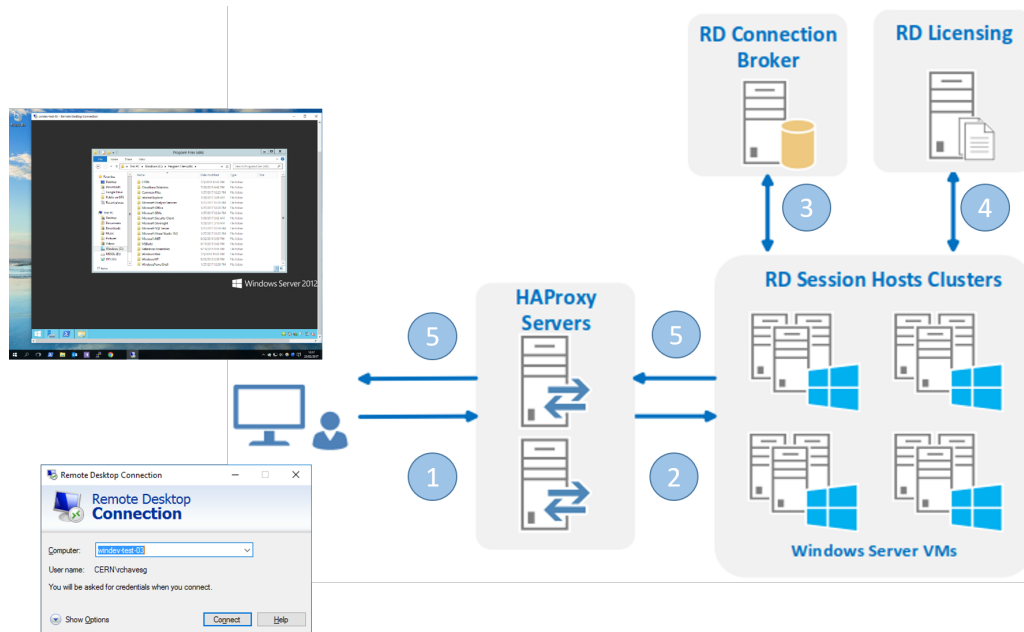


Figure 2.6: CERN Windows Terminal Service connection workflow.

made to RDG (1), which validates if there are licenses available to accommodate a new connection (2) and then establishes the connection (3).

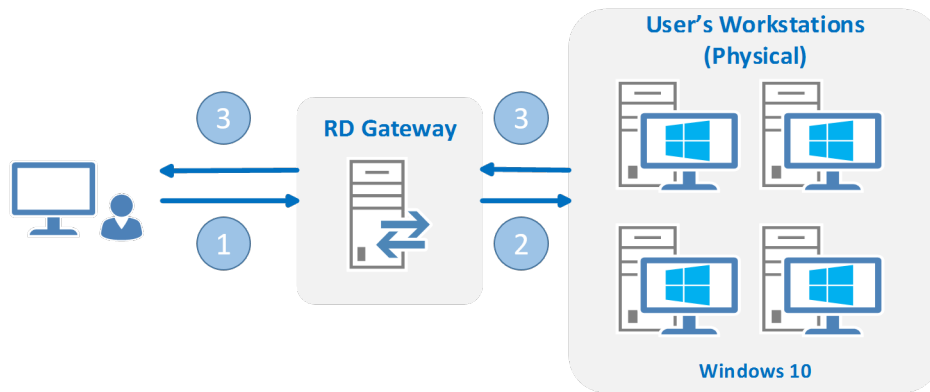


Figure 2.7: Remote Desktop Connection to CERN workstation.

In both scenarios, the only components exposed to the outside network are HAProxy and Remote Desktop Gateway servers. Even though it is not depicted, there is a firewall between the HAProxy/RDG and the external network, for enhanced security.

It is also important to refer why a (Windows) Server version is used for the user-accessible (i.e., running RDSH server role) hosts, instead of the usual desktop versions (e.g. Windows 7,8,10). The reason has to do with the ability to allow multiple Remote Desktop sessions at the same time: only server versions allow them.

Having described both scenarios, we must refer that our work is focused on the infrastructure responsible for remote desktop connections to RDSH servers (first scenario).

### 2.1.3 HAProxy

HAProxy [39, 40] stands for “High Availability Proxy”, an open source software that is both a proxy server and a load balancer that can be run on Linux, Solaris, and FreeBSD. It has been designed for scenarios where availability is an important requirement, i.e., down times should be short (a few minutes, at most). It is capable of load balancing and proxying TCP and HTTP-based traffic and also has monitoring capabilities. It’s used by many high-profile websites including: GitHub, Stack Overflow and Twitter.

HAProxy uses two concepts to deal with the services (e.g., webserver/applications) it mediates access to: back-end and front-end.

- A **back-end** is a set of servers that receives forwarded requests and is defined by the load balancing algorithm to use and a list of servers and ports.
- A **front-end** defines how requests should be forwarded to back-ends. Their definitions are composed by a list of Internet Protocol (IP) addresses and ports, ACLs and back-end rules. A front-end can be configured to various types of network traffic.

HAProxy can be considered a fairly complete load balancer regarding the set of algorithms it supports (9 different load balancing algorithms) [40]. The most common are:

- **round-robin** - for short connections, pick each server in turn;
- **leastconn** - for long connections, chose the least recently used of the servers with the lowest connection count;
- **source** - for SSL farms (groups of servers) or terminal server farms, where the server directly depends on the client’s source address;
- **uri** - for HTTP caches, the server directly depends on the HTTP URI;
- **hdr** - the server directly depends on the contents of a specific HTTP header field;
- **first** - for short-lived virtual machines, all connections are packed on the smallest possible subset of servers so that unused ones can be powered down.

In the WTS infrastructure the *leastconn* algorithm is used to ensure a good load balance of RDS sessions (long connections).

Some applications require that a user that for some reason has been disconnected (without closing the session) will, when reconnecting, be directed to the same back-end server. In HAProxy, this persistence is achieved through sticky-tables. Sticky-tables are commonly used to store stickiness information, that is, to keep a reference to the server a certain user was directed to, using the identifier associated with the user (IP address, SSL ID of the connection, HTTP or RDP cookie, etc.) as a key and the server’s identifier as the associated value [40]. Through the use of sticky-tables in the WTS infrastructure, a user’s

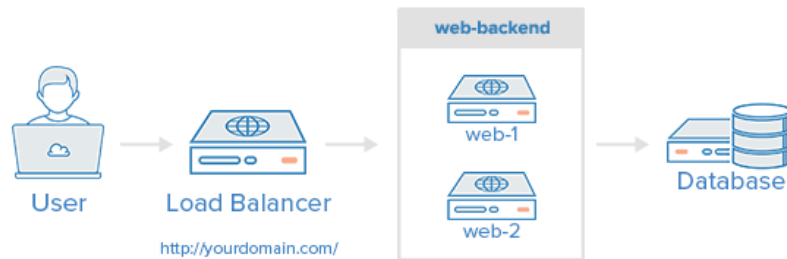


Figure 2.8: Load balancing example showing the back-end concept [41].

RDS session can be recorded, enabling requests (from same client) to be redirected to the same server.

### High Availability for the HAProxy: Keepalived

CERN uses the Keepalived [42] software to monitor services or systems for failures and, upon failure, restarts the failed service/server - in a different node, if needed.

Keepalived uses a floating IP address that can be moved between servers. In the case of two servers, for example, if the primary server goes down, the floating IP will be automatically moved to the second server, allowing it to resume the service with the same IP, thus keeping it transparent to clients [43]. An example diagram showing the usage of Keepalived with two HAProxy servers is depicted in figure 2.9.

CERN Windows Terminal Service has two HAProxy servers monitored by keepalived to provide a highly available and redundant infrastructure, eschewing a single point of failure.

### Feedback Agent

Feedback Agent (FA) [44, 45] is a daemon designed to run on back-end servers that provides a way for HAProxy to get their workload status (CPU and memory usage percentage). HAProxy servers can take into account the server's workloads and redirect traffic to those which are less loaded.

This agent has three modes: normal, down and drain. Normal, means that the server can receive connections; Down, means the server stops all the connections and doesn't accept new ones; Drain, server doesn't accept new connections, current connections are kept until they are terminated.

In the context of WTS, FA is installed on RDSH servers to provide memory usage percentage - as some of the applications used are memory sensitive - to HAProxy so it can load balance using this workload data and also the number of connections (RD sessions) each machine has. Currently HAProxy is configured to redirect connections to other servers when a server reaches 90% of memory used optimal load balancing. The drain mode is useful when the servers are subject to maintenance tasks or need to be replaced, allowing RD sessions to be redirected to other servers.

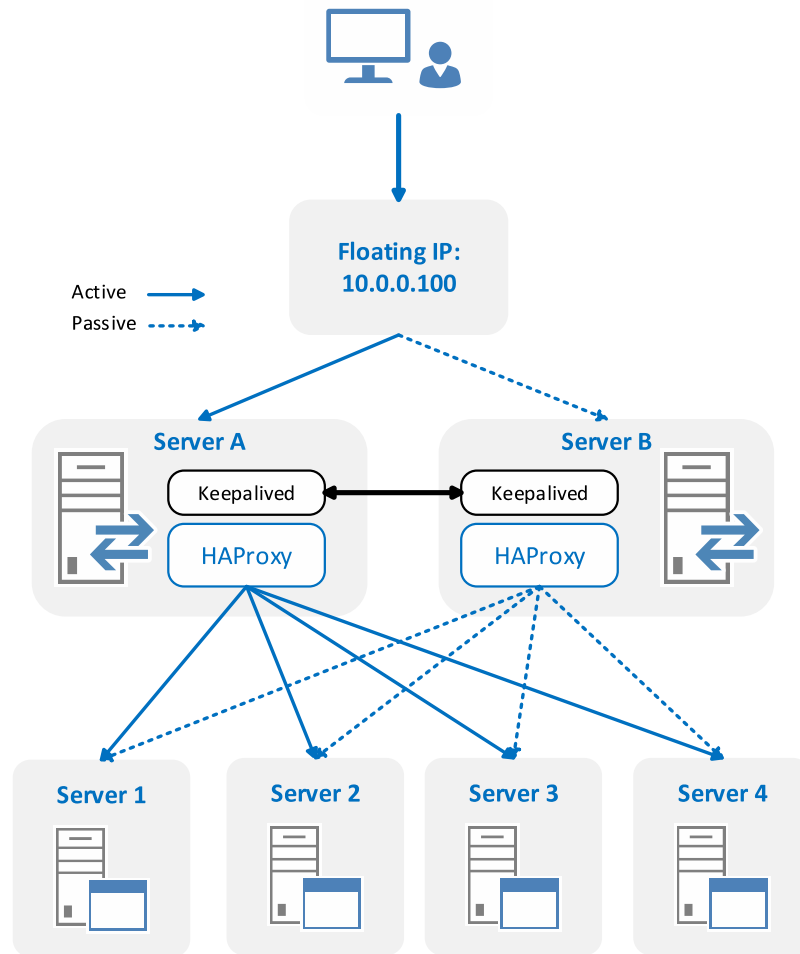


Figure 2.9: HAProxy+keepalived example.

## 2.2 Monitoring the WTS Operations: SCOM

System Center Operations Manager (SCOM) [46] is a comprehensive monitoring solution that offers centralised monitoring and management of applications, virtual environments, physical environments, and cloud-based workloads. SCOM is the core monitoring solution from Microsoft for over a decade and supports both Microsoft Windows Server and Unix-based systems.

SCOM uses a single interface that shows the state, health and performance information of computer systems, and also provides alerts generated in response to events that may signal problems, e.g., in availability, performance, configuration or security.

SCOM is a software solution designed for large scale infrastructures that run critical services and applications whose performance and availability must be ensured. This means that SysAdmins need to know when there is a problem, identify where the problem is, and figure out what is causing it, ideally before it gets noticed by the users. The larger

the scale of the infrastructure, the more challenging this task becomes [47].

In System Center Operations Manager, the administrator can configure what to monitor by selecting targets as diverse as computers, devices, services and applications. Some targets may require importing management packs that provide specific agents for the items being monitored.

A SCOM infrastructure is composed by 4 components [46]:

- **Management Server** - is the central point for administering the management group and communicating with the database.
- **Operational Database** - is a SQL Server database that contains all configuration data for the management group and stores all monitoring data that is collected and processed for the management group. It retains short-term data, by default 7 days.
- **Data Warehouse Database** - is also a SQL Server database that stores monitoring and alerting data for historical purposes. Data that is written to both databases, so reports always contain current data. The data warehouse database retains long-term data.
- **Reporting Server** - is responsible for generating and presenting reports based on the data stored in the data warehouse database.

In WTS, SCOM is used to monitor availability, workload and health status of servers and databases; SCOM alarms are also used to trigger alerts when some resources are reaching critical status (e.g. not enough free disk space in a particular server).

## 2.3 Management and Configuration of the WTS infrastructure

As previously referred, the WTS infrastructure is currently configured and managed using services commonly found in Windows domain networks, such as Active Directory and Group Policies (that help manage computing resources like users, computers and printers) and other mechanisms like scripts and configuration management software tools. In this subsection we briefly introduce these services and mechanisms and explain how they are used in CERN's WTS.

### 2.3.1 Active Directory

Active Directory (AD) [48] is a directory service that Microsoft developed to centralise management of Windows (domain) networks.

Generically speaking, a directory is a listing of objects; an example of a directory is a phone book: it stores information about people, businesses, and organisations. AD stores information about organisations, sites, systems, users, file shares, and many other entities, or objects; these fall into two broad categories: resources (e.g., printers) and



security principals (user or computer accounts, and groups). Each object represents a single entity (user, computer, group, or printer) and its attributes.

AD allows clients to find objects within its namespace - the area in which a network component can be located. Using the previous example, phone books provide a namespace for resolving names to telephone numbers in the same way that DNS is a namespace that resolves host names to IP addresses. In AD, the namespace is used for resolving the names of network objects to the objects themselves. Namespaces can be organised into multiple hierarchical levels: forest, tree, and domain.

A domain is as a logical group of network objects (computers, users, devices) that share the same AD database, while a tree is a collection of one or more domains into a contiguous namespace.

The forest represents the top of the structure and is a collection of trees that share a common global catalog, directory schema, logical structure, and directory configuration. The forest represents the security boundary within which users, computers, groups, and other objects are accessible.

The objects held within a domain can be grouped into Organisational Units (OUs) allowing to mimic an organisation structure and thus simplifying the implementation of policies and reducing administration burden [49, 50].

Typically, to apply and enforce a set of configurations and polices to same OUs or domains, Group Polices are used, as explained in subsection 2.3.2 below.

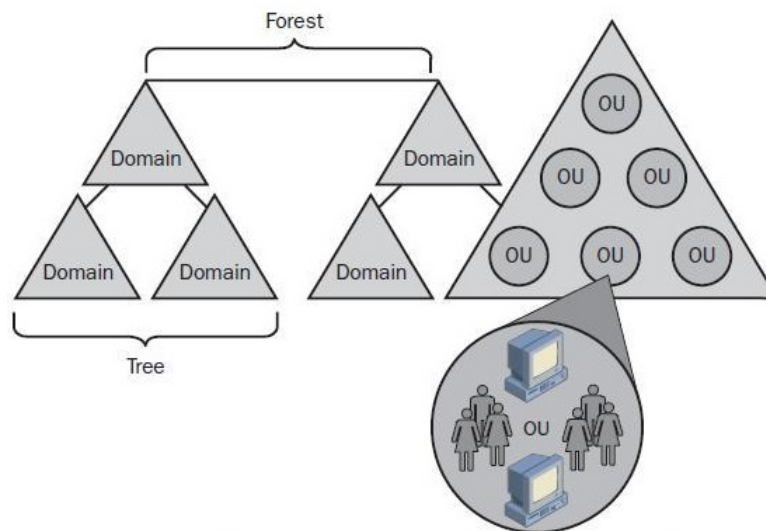


Figure 2.10: Active Directory logical structure [51].

### 2.3.2 Group Policy

Group Policy is a feature of Windows domain networks that provides centralised management and configuration for Windows operating systems, applications and user settings.

Group policies are Active Directory objects formally named Group Policy Objects (GPOs) that contain the rules and settings to be applied to an Organisation Unit (OU) or domain.

If a set of users needs to have the same standard desktop configuration and the same set of applications, the users can be put into an OU, and Group Policy can then be used to configure the desktop and to manage the installation of applications [50]. There are many types of policy settings available; we list a few below:

**Administrative templates** Used to manage registry-based parameters for configuring application settings and user desktop settings, including access to operating system components and to the control panel.

**Security** Used to manage the local computer, domain, and network security settings, including controlling user access to the network, configuring account policies, and controlling user rights.

**Software installation** Used to centralise the management of software installations and maintenance.

**Scripts** Used to specify scripts that can be run when a computer starts or shuts down, or when a user logs on or off.

**Folder redirection** Used to store certain user profile folders on a network server. These folders, such as the My Documents folder, appear to be stored locally but are actually stored on a server where they can be accessed from any computer on the network.

**Preferences** Used to manage options related to Windows settings or Control Panel settings, including drive mappings, environment variables, network shares, local users and groups, services, devices, and many more.

Group Policy allows the SysAdmin to define policies that control software deployment in a very limited form. It is GUI-based, and therefore doesn't provide an automated way of deploying software, or reporting about its installation status.

### 2.3.3 Scripting

Scripting is most basic approach when it comes to automate the deployment and configuration of operating systems. SysAdmins rely on them to do most of the tedious and repetitive tasks that involve multiple steps in order to obtain the desired effects, while avoiding error prone operations.

Generically, scripting languages run on a special run-time environment and offer primitives to operate the underlying system; these primitives are usually elementary tasks or API calls, and the language allows them to be combined into more complex programs.

Until the introduction of PowerShell, Windows SysAdmins had to rely on the system shell and other scripting languages (e.g. JScript [52], VBScript [53]) to automate tasks. However, the system shell cannot be used to automate all facets of the GUI's functionality, in part because command-line equivalents of operations exposed via the graphical interface are limited, and the scripting languages are elementary and don't allow the creation of complex scripts [54].

In 2006, Microsoft introduced the PowerShell automation and scripting language for the Windows platform, one that allows to simplify the management of these systems [54]. Unlike other text-based shells, PowerShell takes advantage of .NET Framework, providing an object-oriented paradigm and a massive set of built-in functionality for taking control of Windows environments.

PowerShell enables scripts to perform virtually any task one can do using the Windows' GUI; it includes numerous system administration utilities, and has consistent syntax and naming conventions, and improved navigation for common management data, such as the registry, certificates stores, and WMI [55] - a core technology for Windows system administration, because it exposes a wide range of information in a uniform manner [56]. As described by Microsoft:

"Windows Management Instrumentation (WMI) is the Microsoft implementation of Web-Based Enterprise Management (WBEM), which is an industry initiative to develop a standard technology for accessing management information in an enterprise environment. WMI uses the Common Information Model (CIM) industry standard to represent systems, applications, networks, devices, and other managed components. CIM is developed and maintained by the Distributed Management Task Force (DMTF)." ([57])

PowerShell provides an interface and programming environment that allows users and administrators to access and set system properties through .NET objects and single-function command-line tools called *cmdlets* which are the building blocks of PowerShell scripts.

A lot of the current automation and configuration effort of WTS is based on PowerShell scripts, due to their flexibility and to the advantages presented above.

### 2.3.4 Computer Management Framework

The Computer Management Framework (CMF) [58] is a software framework, designed and developed at CERN, that offers a range of tools that enables groups of Windows computers to be easily managed. CMF's main functionality is to distribute software applications to computers in a user-friendly and managed way; it replaced the Microsoft Systems Management Server (SMS) [59] tool and some group policies and start-up scripts that were previously used.

CMF runs as an agent in Windows systems so that users can interact and control and schedule deployment processes. The operations are done through a web portal where is possible to manage computer membership and select the software packages to deploy. It

also offers improved security, stricter control on patch deployment and reboot actions, and enables instant reporting.

A domain is called a Name System Set (NSS) in CMF, and the framework is based on the concept of hierarchical delegations between a master NSS (domain) and sub-domains. The master NSS has full control over the whole domain and can create other NSSs and grant them permissions; the idea is to structure NSSs in a way so that each NSS can control a well defined set of computer or user groups. The role for each NSS is defined by the permissions that were granted by the master NSS.

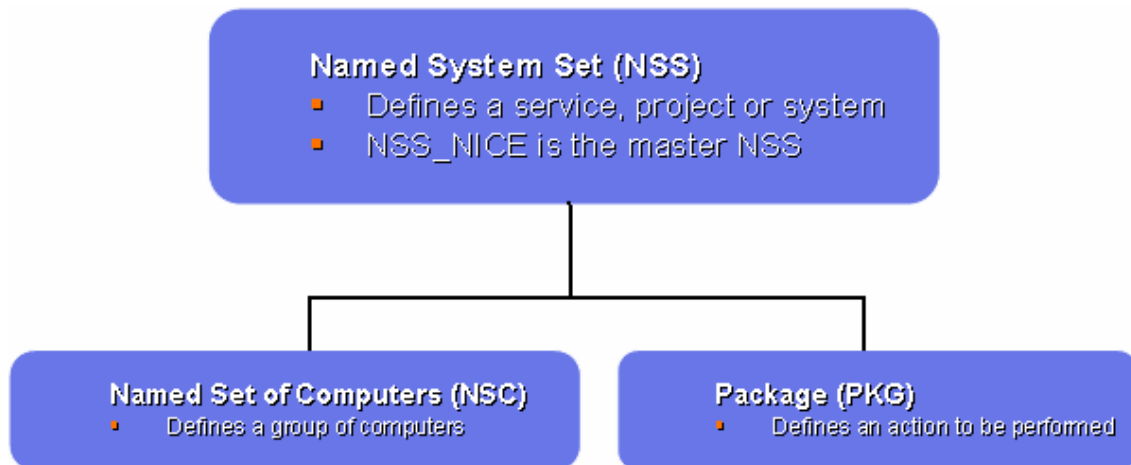


Figure 2.11: Computer Management Framework logical structure [58].

## AGILITY

This chapter is about agility; agility is of paramount importance today, not only in development but also in operations, as only after deployment the new product or version can be used and appreciated.

Introduction of agility in the development process is generally acknowledged to have happened around the year 2000 (although some trace it back into the 70s and Smalltalk) but, as far as infrastructure operations are concerned, that process is very recent.

In this chapter we start by introducing DevOps, which has merged agile software development with agile infrastructure operations and then explain how the later was a result of the adoption of the Infrastructure-as-Code (IaC) paradigm in the new tools that are now in place.

### 3.1 DevOps

After the birth of Web 2.0 (but before cloud services) some IT companies (e.g. Amazon) were facing the problem of managing the configuration of large-scale infrastructures - a problem with two axes: management, and software delivery. Big service providers have different teams responsible for different parts of the services they deliver: development team, continuous integration and testing team, and operations team. Development teams are responsible for software development and their focus is to continuously deliver new features and functionalities; continuous integration and testing teams provide tests and tools to ease the process of integration of new software features and prevent them from reaching production with bugs or erroneous behaviour; operations teams are in charge of managing and configuring the underlying infrastructure where the software will run (i.e. desktop computers, servers, mobile devices).

### 3.1.1 Before DevOps

Before agile development methodologies were adopted, the traditional approach of developing software was based on the Waterfall model [60], which was first formally defined by Winston W. Royce in 1970 - although he didn't use the term "waterfall". In the Waterfall model, the whole process of software development is divided in distinct phases which are carried out sequentially - a phase only starts when the previous one has ended. Phases in Waterfall model are [61]:

1. **Requirement Gathering and Analysis** - All possible requirements of the system to be developed are gathered and documented in a Software Requirements Specification (SRS) document.
2. **System Design** - The requirement specifications from first phase are studied and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
3. **Implementation** - Based on system design, the system is first developed in small programs called units and then integrated in the next phase. Each unit is developed and tested for its functionality, a.k.a. Unit Testing.
4. **Integration and Testing** - All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration, the entire system is tested for any faults and failures.
5. **Deployment** - Once the functional and non-functional testing are done, the product is deployed in the customer environment or released to the market.
6. **Maintenance** - After deployment, usually there are some issues which come up in the customer environment. Hence, patches or new versions are released in order to enhance the system. Maintenance is the delivery of changes to the customer environment.

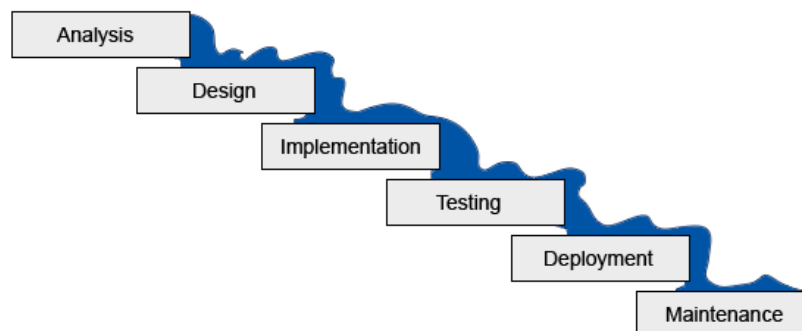


Figure 3.1: Waterfall model [62].

This model is simple and easy to understand and implement: each phase has to be completed in sequence, and has specific deliverables (documents, tests, code); and this approach is quite suitable for small projects where the requirements are very well defined and understood.

But the waterfall model is not suitable for projects whose requirements are subject to frequent change and when the final software/product is not clear; it is also not suitable for projects that require large-scale infrastructures - the waterfall model was not designed to cope with uncertainty and frequent changes: once in a testing phase, is very difficult to go back and change something that was not very well thought in the design phase.

In recent years, big software projects and large-scale infrastructures are more common and it's hard to design and develop software using models that don't cater for unforeseen changes. Adopting such a model would be designing to failure, increasing the costs of such projects. On the other hand, the Agile development model, an approach that results in faster software development, incorporates design changes and relies on small but frequent releases of new features.

However, this approach raises lots of conflicts between the development and operations teams: developers want to deliver new functionalities as soon as they have them, following an Agile development model, while operators resist against new changes as they are afraid of disrupting production services. Operations teams usually have plenty of responsibilities in addition to software deployment, including managing costs, user accounts and overall capacity, and ensuring overall security [6]. Continuous integration and testing teams served as a buffer between the other two groups, thus avoiding some conflicts, but a better solution was the introduction of the DevOps approach, something that we will study in more detail further down.

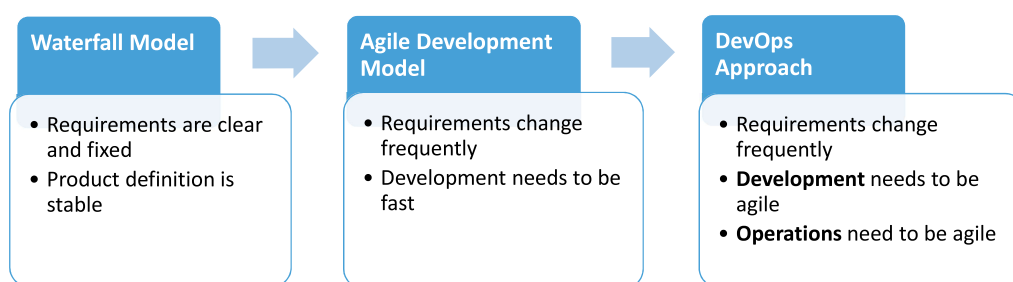


Figure 3.2: Evolution of software development (adapted from [63]).

So, in order to achieve frequent delivery of small software changes, a new set of tools and techniques was needed. The new set of tools that was created was heavily influenced by two concepts: Infrastructure as Code (IaC) and Software Configuration Management (SCM) [6].

IaC is a result of the increase of Software-as-a-Service (SaaS), IaaS (e.g. Amazon Web Services) and APIs that provide functions for the management of large-scale/cloud infrastructures, e.g., that cater for the provisioning of virtual machines/cloud instances.

Combined with SCM tools, the computing infrastructure can be now handled in the same way software is (see 3.2).

The new breed of SCM tools are a result from the application of the IaC methodology and provide a way of describing the desired state of an infrastructure through Domain Specific Languages (DSLs), allowing SysAdmins to both create and manage the infrastructure using the same tools, as described in section 3.3.

In addition, the increasing prevalence of simpler and easier-to-use APIs (like RESTful APIs) has allowed non-developers to use them, resulting in a wider adoption from operators (SysOps) and enabling them to do development as opposed to scripting. This engaged more people into learning basic software engineering practices.

As a response to these needs, a new approach for software development was born and coined as DevOps.

### 3.1.2 What is DevOps?

The term DevOps results from the combination of the words “development” and “operations” and emphasises the collaboration between development and operations processes/tasks. However, a better definition can be found in a recent study carried out by Jabbari et al. [2] which aimed to formally define DevOps based on the available literature:

“DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops), emphasising communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilising a set of development practices.” (adapted from Jabbari et al. [2])

Lwakatare et al. [4] described a conceptual framework that depicts the problems that DevOps aims to solve, its principles and its outcomes. Table 3.1 summarises the pillars on which DevOps is based.

Table 3.1: Conceptual framework characterising DevOps (adapted from [4]).

Problems addressed by DevOps	Principles of DevOps	Outcomes of DevOps
Poor communication	Collaboration	Shared responsibility; one team responsible for entire service or product
Manual operations (e.g. deployment, configuration management)	Automation	Continuous deployment of functionality and infrastructure provisioning
Performance of development and QA are not supported by data	Measurement	Operational data to also measure performance of development
Monitoring data is segregated and voluminous	Monitoring	Consolidated view of operational data as feedback



When relying on a traditional software development model like waterfall, development and operations teams face different challenges for which DevOps offers a solution. Table 3.2 illustrates those challenges.

Table 3.2: How DevOps addresses different challenges (adapted from [64]).

Team	Challenge	DevOps Solution
Development	Waiting time for code deployment	Continuous Integration ensures there's a quick deployment of code, faster testing and feedback mechanism
	Pressure of work on old, pending and new code	No waiting time to deploy the code and developers can focus on building the current code
Operations	Difficult to maintain up-time of the production environment	Virtualization provides a flexible way of provisioning new machines to replace and ensure the production environment uptime
	Tools to automate infrastructure are not effective	Configuration Management helps organise and execute configurations, ensuring compliance and proactively manage the infrastructure
	High and continuously growing number of servers Difficult to diagnose problems and provide feedback	Continuous Monitoring lets keep track the status of the infrastructure

The DevOps approach extends the concepts of the Agile development model, enabling developers increase the rate of software releases and operations teams to automate a substantial part of their work [2, 65]. The fundamental goal is to remove all bottlenecks in the software development process by automating software integration, testing, deployment and infrastructure configuration so that software releases can be performed quickly, frequently and reliably. This approach defines the notion of a dynamic infrastructure, also known as Agile Infrastructure [66, 67], capable of adapting to frequent changes.

DevOps practices, as presented in table 3.3, are a logical consequence of the principles previously referred in table 3.1.

Table 3.3: DevOps practices (adapted from [2]).

Knowledge Area	Sub-Knowledge Area	Practice
<b>Software Engineering Management</b>	<b>Software Project Planning</b>	Continuous planning Feedback loop between developers and operators
	<b>Software Project Enactment</b>	Continuous monitoring Automated performance monitoring during test and continuous integration Automated feedback for performance models and performance predictions Application monitoring Automated dashboards
<b>Software Construction</b>	<b>Practical Considerations</b>	Continuous integration
	<b>Software Construction Fundamentals</b>	Prototyping applications
<b>Software Configuration Management</b>	<b>Software Release Management and Delivery</b>	Integrated deployment planning Continuous deployment Automated deployment Continuous delivery Cooperative application configurations Monitoring application and next development
	<b>Management of the SCM Process</b>	Staging application Integrated configuration management
	<b>Software Configuration Control</b>	Integrated change management Change management
<b>Software Testing</b>	<b>Test Techniques</b>	Continuous testing Automated testing
<b>Software Process</b>	<b>Process Definition</b>	Process standardisation Production support
<b>Software Quality</b>	<b>Practical Considerations</b>	Use of data to guide QA
<b>Software Engineering Tools and Methods</b>	<b>Software Engineering Methods</b>	Infrastructure as code Modeling & Simulation Measure performance metrics Continuous application performance
	<b>Software Engineering Tools</b>	DevOps maturity evaluation model Elasticity practice
<b>Software Requirements</b>	<b>Software Requirements Fundamentals</b>	Defining requirements
	<b>Requirements Process</b>	Stakeholder participation
<b>Software Design</b>	<b>Software Structure and Architecture</b>	Designing architecture

Adopting DevOps in one's organisation requires both cultural as well as technological changes; the latter is supported by a set of tools, the DevOps toolchain, each one addressing a phase in the process (from the development down to the operations):

- Plan - Planning of the software to develop, gather requirements;
- Code - Code development and review, version control tools, code merging;
- Build - Continuous integration tools, build status;
- Test and Verify - Continuous testing tools that help verify and validate the software quality and stability;
- Package - Artefact repository, application pre-deployment staging;
- Release - Change management, release approvals, release automation;
- Configure - Infrastructure configuration and management, Infrastructure as Code tools;
- Monitor - Applications performance monitoring, end-user experience.

Figure 3.3 illustrates these phases (where some "combination" of adjacent phases has been applied for the sake of brevity).

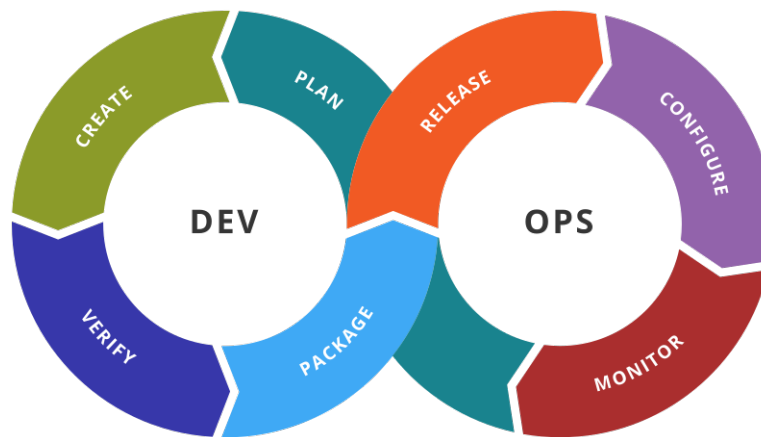


Figure 3.3: DevOps life cycle [68].

These phases in the DevOps life cycle are associated with some of its practices: Continuous Development, Continuous Testing, Continuous Integration, Continuous Deployment/Delivery. One can note a pattern in these terms with the word “continuous”, which reflects the feedback loop - continuous cycle - between stages, a cycle that goes on repeating itself until the desired quality is achieved for the product. For each stage there is a number of tools that can be used to perform the required tasks.

**Continuous Development (CDev):** This is the stage where the software is developed continuously. Unlike the Waterfall model, software deliverables are broken down into multiple sprints of short development cycles, developed, and then delivered, all in a very short time. This stage involves the Coding and Building phases and makes use of tools such as Git [69] and SVN [70] for maintaining the different versions of the code, and tools like Ant [71], Maven [72], Gradle [73] for building/packaging the code into (executable) files that are forwarded to the testing stage.

**Continuous Testing (CT):** The stage where the software already developed is continuously tested for bugs. For Continuous testing, test automation tools like Selenium [74], JUnit [75], etc. are used. These tools enable QA testing of multiple code-bases thoroughly in parallel to ensure that there are no functionality flaws. In general there are at least three environments to run and test software: Development - for the development stage; QA or Test - to test the code; and Production - the final environment where the software is running. The best practice is to have four [76, 77], one more, usually called Staging or Pre-production, inserted before production, an environment very similar to the production's, used to perform final tests before moving to production. But more environments can be used [78]. In this phase, the use of Docker [79] containers to simulate the test environment, on the fly, is also a preferred choice. Once the code is tested, it is continuously integrated with the existing code.

**Continuous Integration (CI):** The stage where the code supporting new functionality is integrated with the existing code. Since there is continuous software development, the updated code needs to be integrated continuously as well as smoothly with the current systems in order to reflect the changes onto the end users. The modified code should also run with no runtime errors in the new environment, allowing us to test not only the changes made to the code but also their interaction with the runtime environment. One of the most popular tools used for Continuous Integration is Jenkins [80]. Using Jenkins one can pull the latest code revision from Git repository and produce a build which can finally be deployed to test or production server. It can be set to trigger a new build automatically as soon as there is change in the Git repository or triggered manually on click of a button.

**Continuous Deployment/Continuous Delivery (CD):** This is the stage where code is deployed to the production environment. The most basic requirement is to ensure that the code is correctly deployed on all servers. If there is any new functionality or a new feature is introduced, then one should be able to withstand an increase in the resource usage (e.g., server communication traffic). So, it is also a responsibility of the Operations team to scale up the servers to host more users. Since new code is deployed on a continuous basis, automation tools play an important role as the above-mentioned tasks should be executed

quickly, and frequently. Puppet, Chef and Ansible [81] are some popular tools that are used in this stage.

**Continuous Monitoring (CM):** This is a very crucial stage in the DevOps life cycle, aimed at improving the quality of the software by monitoring its performance, something that requires the participation of the Operations team, that will monitor user activity for bugs and any unusual behaviour of the system. Usually, the use of application-level monitoring that complement the basic system performance statistics is important, and there are dedicated tools that perform those tasks and highlight any issues. Some popular tools used are Nagios [82], NewRelic [83] and Sensu [84]. These tools help you monitor the application and the servers closely to check the health of the system proactively. They can also improve productivity and increase the reliability of the systems, reducing IT support costs. Any major issues found to be in the code's realm could be reported to the Development team so that they can be fixed in the continuous development phase.

## 3.2 Infrastructure as Code

Infrastructure as code (IaC) is an approach to infrastructure automation based on practices from software development. It emphasises consistent, repeatable routines for provisioning and changing systems and their configuration [1].

The premise is that infrastructure can be treated as if it were software and data. This allows to manage infrastructures applying the same principles, practices and tools used in software engineering: Version Control Systems (VCSs), Test-Driven Development (TDD), Continuous Integration (CI), Continuous Delivery (CD), automated testing libraries, and deployment orchestration. Hence system administrators (Operations team) can benefit from the use of these mechanisms to leverage automation across their organisation's infrastructures.

IaC was introduced with cloud infrastructures as a result of the difficulties of managing servers using traditional configuration and deployment strategies. Nevertheless, the principles and practices of IaC can be applied to any infrastructure independently of its nature: physical, virtualised, or cloud-based.

Cloud providers rely on virtualisation platforms to take the most out of the available hardware and create flexible infrastructures; and IaC enables the concept of a dynamic infrastructure by providing the ability to create and destroy servers programmatically.

But even hardware can be automatically provisioned so that it can be used in a fully dynamic fashion - sometimes referred as a "bare-metal cloud" [1]: it is possible to use many of the concepts of infrastructure-as-code with a static infrastructure. Furthermore, servers that have been manually provisioned can be configured and updated using configuration management tools.

However, the ability to effortlessly destroy and rebuild servers is intrinsically related with many of the principles of IaC.

### 3.2.1 Principles of IaC

Morris [1] laid out IaC's set of principles, and we shall present them as follows:

**Systems Can Be Easily Reproduced** The possibility to effortlessly and reliably rebuild any computer in an infrastructure. Effortlessly means that there is no need to make any significant decisions to rebuild an element of the infrastructure. Decisions related with software versions to install, the hostname selection, and others should be captured in the scripts and configuration tools that provision it. The ability to effortlessly build and rebuild any part of the infrastructure removes much of the risk and fear when making changes, since failures can be handled quickly and with confidence. It also allows the provisioning of new services and environments with little effort.

**Systems Are Disposable** The benefits of a dynamic infrastructure reside on its disposability: resources can be easily created, destroyed, replaced, resized, and moved. In order to take advantage of this, systems should be designed with the assumption that the infrastructure will always be changing. A dynamic infrastructure allows to make live (runtime) improvements and fixes and that, in turn, makes services more tolerant to failure, something that is very important in large-scale/cloud infrastructures, since the reliability of all of the underlying hardware can't be ensured.

**Systems Are Consistent** Having consistent infrastructure means that systems which provide similar services should be nearly identical (e.g. application servers in a cluster that provide the same service). In those systems, the software and configuration should be the same with the exception of some configuration items that differentiate them and make them unique (e.g., IP addresses). The existence of inconsistent systems (in the same cluster that provide the same service) makes its automation more difficult since systems that don't quite match (different characteristics) increase the configuration's complexity. This principle relies on reproducibility: systems should be built using the same procedure to ensure its similarity and consistency.

If, for example, there are two file servers and one of them needs to be changed (e.g. adding a larger disk partition), there are two ways to keep consistency:

1. Change the definition so that all file servers are built with a large enough partition to meet the need.
2. Create another configuration definition, one that differs from the standard one by having a larger disk.

Hence, either type of server can be built repeatedly and consistently.

**Processes Are Repeatable** Also in accordance to the reproducibility principle, any action carried out on the infrastructure should be repeatable; therefore, scripts and configuration management tools should be used instead of making changes manually, although it can be hard, especially for experienced system administrators. Even if it takes some time to develop and test such scripts and use configuration management tools, the long-term benefits of automating processes are obvious. Besides, teams need to rely on reproducible automated processes to ensure that all team members perform these actions in the same way, adhering to policies, rules and best practises, and therefore avoiding configuration drifts caused by human interaction.

Effective infrastructure teams have a strong automation culture: automate whenever it's possible. Thus, teams can rely on the processes without worrying about which member knew how to solve a specific problem or configure a specific element.

**Design Is Always Changing** Before cloud infrastructures, the traditional approach of designing a system didn't incorporate change in the design process, thus making it difficult and expensive to change it. Limiting changes to the system once it was built made sense. On the other hand, that approach requires comprehensive initial designs that take "every" possible requirement and situation into account.

Because it's very difficult to accurately predict how a system will be used in practice, and how its requirements will change over time, this approach naturally creates overly complex systems. At the same time, this complexity makes it more difficult to change and improve the system, which makes it less likely to perform well in the long run.

With cloud (dynamic infrastructures), making a change to an existing system can be easier and cheaper. However, software and infrastructures must be designed as simple as possible to meet the current requirements and facilitate change. Changes should be easily, safely and quickly delivered, because they can happen frequently.

As a result, this motivates everyone involved to learn good habits for managing changes, develop efficient and streamlined processes, and use adequate tools.

### 3.3 Configuration Management

Software applications and services require a physical or virtual environment on which they can be deployed and run. Typically, the environment is an infrastructure comprising both hardware and operating system on which software can be deployed. Software applications are decomposed into multiple services running on different servers, either on-premises or in the cloud; while each service has its own application and infrastructure configuration requirements.

In order to deliver software systems to customers (end-users), both infrastructures and applications need to be properly configured. In addition, to prevent service failures and downtime, configuration drifts need to be handled.

Nowadays, software development models like agile require multiple stages and environments, with different configurations, on which an application needs to be deployed, tested and executed. Thus, it is important to ensure that the application can be deployed to multiple environments without undertaking any manual changes to its configuration. In short, the three main challenges that Configuration Management solves are:

- **Large-scale deployment** - configure and deploy applications to a huge amount (hundreds or thousands) of servers;
- **Configuration across different environments** - when migrating from test to production environment, software does not work due to distinct environment configurations;
- **Application version roll-back** - roll-back to a previous stable version of an application when an updated version causes errors.

For this, Configuration Management provides a set of processes and tools which help:

- Ensure that each environment and application has its own configuration;
- Keep track of configuration items;
- Define the relationships between configuration items and how changes in one configuration item will impact another.

Software Configuration Management (SCM) tools achieve this by enabling the configuration at different levels: the infrastructure and the application.

By following IaC principles, SCM tools help in the process of configuring the servers in the infrastructure (OS level) by having configurations described as code and stored in a repository with version control. Therefore, it allows to have the code (configuration) in one place, only accessible by authorised team members who can make changes to it, track those changes and its authorship. Thus, increasing the collaboration and communication between team members. Besides, it's always possible to deliver configurations to new servers ensuring a consistent configuration state across the infrastructure.

The deployment and configuration of applications is the step that follows after provisioning the infrastructure. Applications can be installed and configured in an automated manner by having a central repository or a file server with the application artefacts (e.g. executables, binaries, configuration files, etc.) and the instructions to deploy them programmatically.

In general, configuration settings should be monitored to trace the applied configuration settings, check its status and detect configuration drifts and errors. Therefore, reporting is a key feature of these tools since it helps to verify and ensure that the target systems have the desired state configuration. Furthermore, SCM tools are capable of detecting a configuration drift and re-apply the configuration, and thus ensuring that all the target systems stay compliant with the desired configuration.



## CONFIGURATION MANAGEMENT

Software Configuration Management (SCM) Tools are fundamental to increase the level of automation of an infrastructure. For the WTS, an SCM tool is required to automate the configuration of the Operating System settings (see fig. 1.1). In this chapter, we briefly introduce some concepts and principles of their operation, and present some of the most popular (w.r.t. production use) tools and their principles of operation.

### 4.1 SCM Tools: an Introduction

Let's start by explaining why using these kind of tools is better than scripts. When using scripts to apply configurations SysAdmins face some challenges: the development is hard and takes a lot of time. Typically, scripts are quite verbose and sometimes can be difficult to understand, and that makes its development a hard and time consuming task. Besides, it's not possible to **specify** dependencies between different configuration scripts nor idempotency. Dependencies are useful to indicate configuration settings that should execute based (depending) on the result of the execution of other configurations. Idempotency means that once the desired state is achieved, future executions of the configurations will not produce any changes.

Furthermore, SCM tools typically use a more declarative approach based on DSLs which makes the development of configurations easier by enabling SysAdmins to describe the desired configurations in a declarative manner (without having to be skillful developers). By design, these tools are idempotent, ensuring that configurations are only applied only when needed. They also allow to specify dependencies between configuration settings so that a certain setting will only be executed after others that must precede it have executed successfully. The execution order is calculated at compilation time, based on the dependencies specified in the configuration file.

Another important concept is the architecture that SCM tools adopt: in general, most use a client-server (or agent-master) architecture and therefore usually require an agent (client) installed on the target machines to communicate with the server. As we'll see later on, there are some tools that don't require agents (they are agentless) but still follow a client-server architecture since they use standard protocols like SSH to invoke commands in the target machines.

In a client-server architecture there are essentially two models of delivering configurations to the target machines: **push** and **pull** (see fig. 4.1). In a push-based model, the server sends configurations to the target machines, whereas in a pull-based model, the target machines (the clients) are the ones which fetch configurations from the server. Agent-based tools also allow configurations to be applied in a standalone manner by having the configuration file in the target machine and executing it locally.

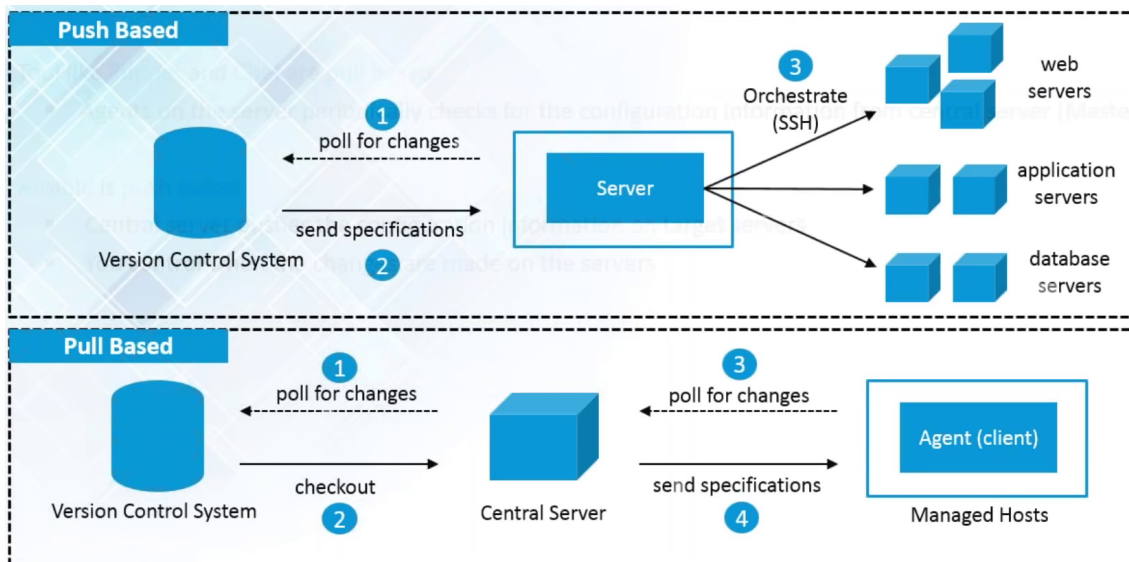


Figure 4.1: Push and pull models [85].

In summary, configuration settings are stored as code (in some DSLs) in a file; that code is then executed in the target machines to apply the correspondent configuration settings.

The execution of the configuration code is usually called a run and it's composed by multiple phases:

1. Gather facts - do an inventory of the system resources (OS, CPU, IP address, etc.);
2. Apply configurations - Check and apply configuration settings (idempotency);
3. Report results - Report to the server the changes made.

The first phase, gathers the facts - information about the system resources of the target machine - which are then fed to the second phase. Then, a local-system specific

configuration code is generated based on the target machine facts. Hence, the configuration code can be identical to a set of machines with different specifications (e.g. different OSs version). After applying the configurations, the run concludes by sending a report of the changes made in the target host, to the server, which gathers all the reports. Reporting is an important feature of SCM tools since it provides a summarised view of the configuration status of the managed infrastructure.

## 4.2 SCM Tools: a brief survey

This section addresses the SCM tools whose primary objective is to manage system configuration in large infrastructures. Notice that one of them (Puppet) is already in place at CERN, as a result of the Agile Infrastructure project, and is used as the main SCM tool.

### 4.2.1 Overview

There are several configuration management technologies currently available that can be used to automate configuration deployment and ensure compliance across the infrastructure; we will now briefly describe some of them, chosen among those that are capable of configuring Windows systems, since this is the main platform of the WTS infrastructure. Despite their differences, all tools have essentially the same goal: manage and deliver configurations to target hosts.

**Puppet [8]** is an open source configuration management solution, built on Ruby, that offers a custom DSL called Puppet Language to create configuration files, using a declarative programming paradigm.

Puppet uses an agent-master architecture where agents manage nodes and request relevant configurations from the master. It allows configurations to be delivered to infrastructure nodes and reports about their status to be retrieved; a more detailed explanation of Puppet can be found in the subsection 4.2.2.

**Ansible [81]** is an open source tool (acquired by RedHat in 2015 [86]) developed to simplify configuration management tasks. It is written in Python and allows users to define configurations in YAML [87] using a declarative programming paradigm. YAML is a human friendly data serialisation language, similar to JSON, designed to work with other languages.

Ansible uses a push model to send, via SSH, commands and configurations to target nodes. It uses an agentless architecture, and the server (a.k.a control machine) is used to tunnel commands and apply configurations to target machines [88].

**PowerShell Desired State Configuration [10]** is a PowerShell-based declarative platform used for the configuration, deployment and management of computer systems [10]; it has a client-agent architecture and supports both the push and pull models.

DSC offers a set of PowerShell language extensions, new PowerShell commands (a.k.a. cmdlets), and resources that can be used to declaratively specify how the software environment must be configured.

Previous to CERN's Agile Infrastructure project, Quattor was the used as the SCM tool for Linux hosts but it was neither agile nor scalable enough for large scale deployment and didn't support Windows hosts. A replacement tool had to be found and, at that time, the choice had to be between the two leading tools: Puppet and Chef [9]. Puppet was chosen mostly as a result of its declarative approach, when compared to the imperative paradigm of Chef. Both were mature, well integrated with other developments, and had good and active communities behind them [15].

However, Puppet approach was similar to Quattor, since they both use declarative languages, and it was considered a better fit for CERN's work processes and for the large number of distinct administrative groups that the tool needed to support.

For these reasons, and in addition to Puppet, we focus our study on Ansible and PowerShell DSC since they also offer a declarative paradigm and have good support for Windows systems.

## 4.2.2 Puppet

### Introduction

Puppet is a configuration management tool that allows a user to define, in a declarative language, a configuration (a manifest, in Puppet's terminology) for a set of resources, specifying the resource's desired states. Then, Puppet enforces that configuration across multiple targets, both hardware or software - such as computer nodes, routers/switches, operating systems, applications, etc.

Configurations are expressed as resources that model complex relationships using Puppet's DSL. A large set of native resources for modelling the desired state of a system is already available; these include the management of, e.g., users, groups, packages, and services.

Puppet uses a declarative language which means that configuration files (a.k.a. manifests) specify the desired state and its attributes (e.g. install and configure a software package, ensure a service is running or manage local users and groups). In short, manifests declare what the state of their hosts should be: what packages should be installed, what services should be running, and so on; System administrators don't need to care about how this state is achieved.

Since version 3, Puppet extended its support to the Windows platform making most of the existing language resources usable to Windows hosts; this survey is based on version 5 (we have currently deployed version 5.1).

In Puppet, there are essentially two ways of applying configurations: to a local machine, in a stand-alone fashion, or to many nodes, using a pull model - the paradigm we use in our work.

In a client-server deployment, a Puppet Master node is used to deploy configurations to client nodes, each running a Puppet Agent daemon [89]. In this model (see fig. 4.2, Puppet Master contains the configuration required for the specific environment. Each agent connects to the Puppet Master through an encrypted and authenticated connection using Secure Sockets Layer (SSL) protocol and fetches the configuration to be applied. As previously mentioned, Puppet configurations are idempotent: if the master has no configuration available for the agent or if the configuration has already been applied, the agent will do nothing - Puppet will only make changes if they are required.

Note that the agent runs with administrative privileges in order to make changes to the target system (on Windows it runs as the SYSTEM user account). Usually, the agent is configured to periodically check with the master to confirm that its configuration is up-to-date or to retrieve any new one. Nevertheless, it's possible to trigger a Puppet run manually from the agent.

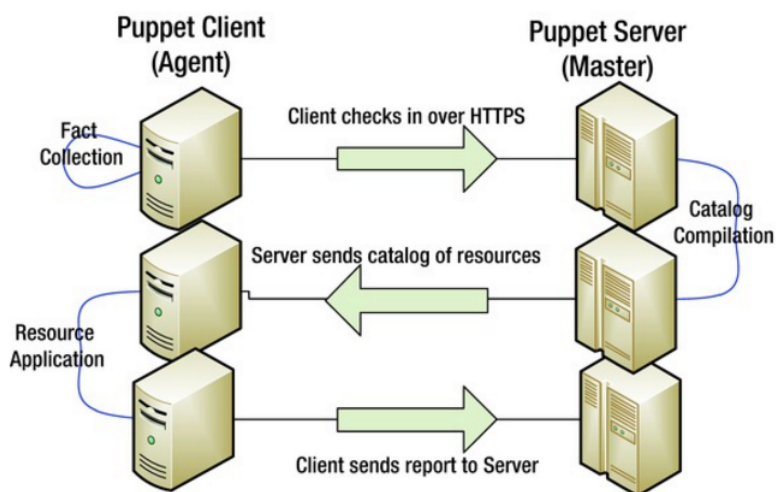


Figure 4.2: Puppet client-server model [90].

Basically, to apply a given configuration, Puppet Master has to:

1. Interpret and compile the configuration stated in the manifest;
2. Communicate the compiled configuration (a.k.a. catalog) to the agent;
3. Apply the configuration on the agent;
4. Report the results of that application to the master.

First, the Puppet Master analyses the configuration in the manifest file and calculates how to apply it to the agent based on its facts. To do this, Puppet creates a dependency graph with all resources declared in the manifest and their relationships. Relationships

between resources can be specified in the manifest, e.g., reboot after changing the page file size. This allows the Master to sort out the order in which to apply each resource to the agent. Puppet then takes the resources and compiles them into a catalog for each agent; the catalog is sent to the Puppet Agent, which then applies the changes; results of this process are then sent back to the master as a report.

### **Puppet Language: the basics**

The Puppet language is all about declaring resources and its attributes. A resource is an abstract concept that can represent things like files, software packages, services, and others. Groups of resources can be organised into classes and be described in manifest files (that end with a .pp extension). A class can describe configurations for a specific application (e.g. Apache Web server) or even an entire service (e.g. configuration settings for a set of front-end Web servers). Classes can also contain other classes to describe system roles or services e.g., “database server”, “application worker”, etc. By default, the order of execution of the configuration items present in manifest files follows the line sequence order. But Puppet allows to specify relationships between resources so that when the catalog is compiled these dependencies may change the execution order.

In Puppet, nodes (devices running puppet agent) can be organised in different groups so that nodes in the same group have the same configurations (classes). The Puppet Enterprise version has a special tool for this purpose called node classifier [91–93]. Since CERN uses the community version, the node organisation is accomplished using another open source tool, called Foreman, that integrates with Puppet as described later. Foreman refers to this organisation as “hostgroups”. From this point on, we will use this naming convention.

Basically, there are two kinds of manifests:

- **Hostgroup manifests** - are aligned with the hostgroup structure (folder tree) and describe the configurations to apply to all nodes in that hostgroup.
- **Module manifests** - are re-usable units of code, each typically configuring one OS daemon or feature, and may be regarded as a library of functions to be used in hostgroup manifests.

Note that Puppet modules are installed on the Puppet Master(s) and the agents only fetch and store their resources (module’s code) according to the hostgroup manifest they must execute.

Listing 1 shows an example of a hostgroup manifest, encapsulated in a class, with configurations for Windows servers. It displays five kinds of resources: **file**, **group**, **service**, **exec** and **registry**. The first four are part of Puppet standard library, whereas the latter is part of the registry module designed for Windows systems.

In Puppet, each resource is declared by its name - i.e. file, group, service, exec -, a title (string between the first brace “{” and colon “:”) and a list of attributes and the

corresponding values. The title of resources of the same type must be unique in order to distinguish them; otherwise an error occurs when compiling the catalog. In some resources, the title string can be used as the default value for the *namevar* attribute (a mandatory attribute) [94]. This attribute is usually the *name* attribute, but in some resources, like *file*, the *namevar* defaults to the *path* attribute. By using the provided string in the title field, it avoids the explicit declaration of that attribute. *File* and *service* resources, in the example manifest, illustrate this well: the title for *file* resource specifies the path for the file to be created, whereas in *service* resource it specifies the name of the service to be managed.

Each resource has specific attributes, but some, like *ensure*, are common to several distinct Puppet resources. This attribute is used to specify the behaviour of the managed resource, and is usually set to either “present” or “absent”, meaning that the resource must either exist, or not, in the target nodes. Depending on the resource type, other values and behaviours are allowed, e.g., the *service* resource uses “running” and “stopped” values to describe whether a service should or shouldn’t be running; whereas a *file* resource allows the “present”, “absent”, “file”, “directory” and “link” values to specify if a file should or shouldn’t be present, create a file, a directory or a symlink (symbolic link - a shortcut to another location in the filesystem).

The resources displayed in the example manifest of listing 1 declare different configuration settings: the *file* resource specifies that a text file with the name “foo” must be created under the specified directory, and should contain the text indicated in the contents attribute; in the *group* resource, the user “rchavesg” should be a member of the local user group named “Power Users”; and, in the *service* resource, the service “WSearch” should be stopped and should not be enabled at boot time.

The *exec* resource is used to execute a given command in the target system; Puppet handles the command as a string, it is not aware of the command language syntax. The *exec* resource has some useful attributes like *provider* that specifies what Command-line Interface (CLI) interpreter should be used to run the command, and *unless*, an attribute that allows us to control whether or not the main command should be run, based on the exit code of the command indicated in this attribute - it executes it only if the exit code is non-zero. In the example manifest, this resource is used to execute a command that sets PowerShell (Windows CLI) to the “unrestricted” mode, allowing it to execute non-signed (i.e., potentially unsafe) scripts. Commands specified both in *main* and *unless* are executed by PowerShell since that is the chosen provider/interpreter.

Finally, the *registry* resource - a Windows-specific module - is used to set values in the Windows registry of the target machine for a specific key. In this case, it will disable User Account Control - a security feature of Windows systems that prevents unauthorised changes to the OS by prompting for administrator-level credentials when an attempt occurs.

```
1 class hg_windows {
2
3   # Create file foo.txt inside C:\Temp\ with some text
4   file { 'c:\\Temp\\foo.txt':
5     ensure => present,
6     content => 'This is some text in my file'
7   }
8
9   # Add user rchavesg to Power Users group
10  group { 'add my user':
11    name   => 'Power Users',
12    ensure => present,
13    members => ['domain\\rchavesg']
14  }
15
16  # Stop WSearch service
17  service { 'WSearch':
18    ensure => stopped,
19    enable => false
20  }
21
22  # Set Powershell Execution Policy to unrestricted
23  exec { 'Set PowerShell execution policy unrestricted':
24    command => 'Set-ExecutionPolicy Unrestricted',
25    unless  => 'if ((Get-ExecutionPolicy -Scope LocalMachine) -eq "Unrestricted")
26              { exit 0 } else { exit 1 }',
27    provider => powershell
28  }
29
30  # Disable User Account Control (change registry key value)
31  registry::value { 'Disable UAC':
32    key   => 'HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System',
33    value => 'EnableLUA',
34    data  => '0',
35    type  => 'dword'
36  }
37
38 }
```

---

Listing 1: Puppet (hostgroup) manifest example for Windows systems.

Listing 2 depicts another manifest example, this one shows other feature set of the Puppet Language. This manifest does essentially two things: changes the page file size of the Windows host and installs a service (a.k.a. server role or server feature).

As shown in the manifest, Puppet supports variables and embedding them inside strings (used in context of the *exec* resource). Variables start with dollar symbol and are followed by a string name (e.g. `$pagefile_initialsize`). Also, it's important to mention that Puppet makes a distinction between single-quoted (e.g. `'a single-quoted string'`) and double-quoted strings (e.g. `"a double-quoted string"`) [95]. Only double-quoted strings allow interpolation - the replacement of variables or expressions for their values. Furthermore, to use some literal symbols (e.g. `$` or `\`) inside the string without being interpolated, they need to be escaped with the backslash symbol (`"\"`).



The *exec* resource also supports the *onlyif* attribute, which is similar to the *unless* but with the opposite behaviour: the main command is executed only if the exit code of the attribute's command is zero.

After changing the page file (using the *exec* resource), the machine must be rebooted; that's why there is a *reboot* resource, one that makes it possible to declare when the host must be rebooted (using the *apply* attribute) and display some text message to any logged-in user (using the *message* attribute). The *apply* attribute accepts "immediately" and "finished" as values to convey whether the restart should happen immediately or if the resource should wait for all the remaining resources in the manifest. In order to trigger the computer to restart only when the page file is changed, the *subscribe* relationship attribute is needed; it declares that it is dependent on the *exec* resource that changed the page file by referring to the resource using its title.

The second configuration in the manifest demonstrates how to install a server feature. Since target machines can have different Windows Server versions and the name of that server role may also be different among them, an *if* statement is used to account for these two scenarios. This is done by storing the appropriate name in a variable (`$search_service`) for latter use.

Puppet DSL is not purely declarative and provides some conditional statements like *if* and *case* [96]. In the example manifest, to compare between different OS, the variable *operatingsystemrelease* - a facter variable - is used. This is an example on how facts can be used in manifests to deliver target system-based configurations. As previously explained, facts are first sent by the agent when it contacts the Puppet master; then the catalog is compiled and sent to the agent, thus delivering the appropriate configuration for each target system.

The *windowsfeature* resource is a Windows-specific module that enables the installation and removal of Windows server roles, controlling it through the *ensure* attribute using the values "present" or "absent". In this example, the server role to install is the Windows search service, whose name is stored in the `$search_service` variable.

```
1 class hg_windows_dev {
2   # Page file sizes
3   $pagefile_initialsize = 2048
4   $pagefile_maximumsize = 4096
5
6   # Set page file using PowerShell commands
7   exec { 'Set-pagefile':
8     command => "Get-CimInstance -ClassName Win32_ComputerSystem |
9       Set-CimInstance -Property @{ AutomaticManagedPageFile = \${false} };
10      Get-CimInstance -ClassName Win32_PageFileSetting |
11      Set-CimInstance -Property @{ InitialSize = $pagefile_initialsize;
12                                MaximumSize = $pagefile_maximumsize }",
13     onlyif => "\$AutoManaged =
14       (Get-CimInstance -ClassName Win32_ComputerSystem).AutomaticManagedPagefile;
15       \$PageFile = Get-CimInstance -ClassName Win32_PageFileSetting;
16       if(\$AutoManaged -or \$PageFile.InitialSize -ne $pagefile_initialsize -or
17       \$PageFile.MaximumSize -ne $pagefile_maximumsize)
18         {exit 0}
19     else
20       {exit 1}",
21     provider => powershell
22   }
23
24   # Reboot after setting page file parameters
25   reboot { 'reboot after pagefile':
26     apply => finished,
27     subscribe => Exec['Set-pagefile'],
28     message => 'Puppet changed the page file settings. This computer will be rebooted now.'
29   }
30
31   # Install and disable search
32   if ($::operatingsystemrelease == '2008 R2') {
33     $search_service = 'FS-Search-Service' # For windows Server 2008 R2
34   } else {
35     $search_service = 'Search-Service' # For windows Server 2012 R2
36   }
37
38   windowsfeature { "${search_service}":
39     ensure => present
40   }
41 }
42 }
```

---

Listing 2: Another Puppet (hostgroup) manifest example for Windows systems.

### Puppet platform and tools at CERN

CERN has deployed a configuration management infrastructure based on Puppet and complemented with other tools [97]; the list is presented below:

- **Puppet:** The open source server software and client daemon responsible for configuring systems. CERN uses the open source edition of Puppet and has recently updated the version to 4.9.4.
- **Facter:** An inventory tool, part of Puppet, that provides information (called facts) about the nodes to Puppet (e.g. IP address, operating system version, CPU model,

uptime, etc.).

- **PuppetDB:** A database that collects all the information from Puppet runs. It can be used to externally query the state of the nodes, or within a configuration to configure dependencies between nodes.
- **Hiera:** A Puppet built-in key/value lookup system with an hierarchical search path that enables the separation code and data. Configurations (written in manifest files) can be modular, so different (hierarchic) hostgroups can be configured based on their data files (specified separately). When Hiera methods are used in the code, Hiera can retrieve on-the-fly (at compilation time) the data (e.g. configuration variables, Puppet modules to be imported) associated with that hostgroup. These data files can be written in YAML or JSON and are stored (and versioned) alongside with manifests (but in a different directory) using the git versioning system.
- **Foreman:** The web front-end for the configuration management system that is used to assign individual hosts to clusters (called hostgroups) and to visualise the reports returned from Puppet runs. It uses PuppetDB to fetch puppet data (runs and facts) and provides dashboards with statistics about the status of the managed machines, thus providing monitoring. Foreman also enables the management of the configuration code (repository) and development environment each machine should look for when fetching configurations. Foreman is not part of the Puppet suite.
- **GitLab:** A platform that provides git repositories to store the puppet code to be applied to the nodes. Gitlab is also not part of the Puppet suite.

An overview of how these components interact with each other is described by the following steps, taken, as an example, as a node gets some service deployed:

1. All hosts in the configuration system are registered in the Foreman service, whose primary function is to state which hostgroup (or cluster of machines) a host belongs to.
2. When a node (“example.cern.ch” depicted in the figure) wants to configure itself, the Puppet Agent on that node will ask the Puppet Master for its configuration. The first thing the Puppet master will do, is to ask the Foreman service for what hostgroup the node belongs to.
3. Puppet Master will find out what configuration needs to be applied to the node in that hostgroup (e.g., what packages and configuration are needed to deploy a “web-server/basic”). It does this by reading, from a Git repository, the Puppet manifests (code) and Hiera (data) for that hostgroup.

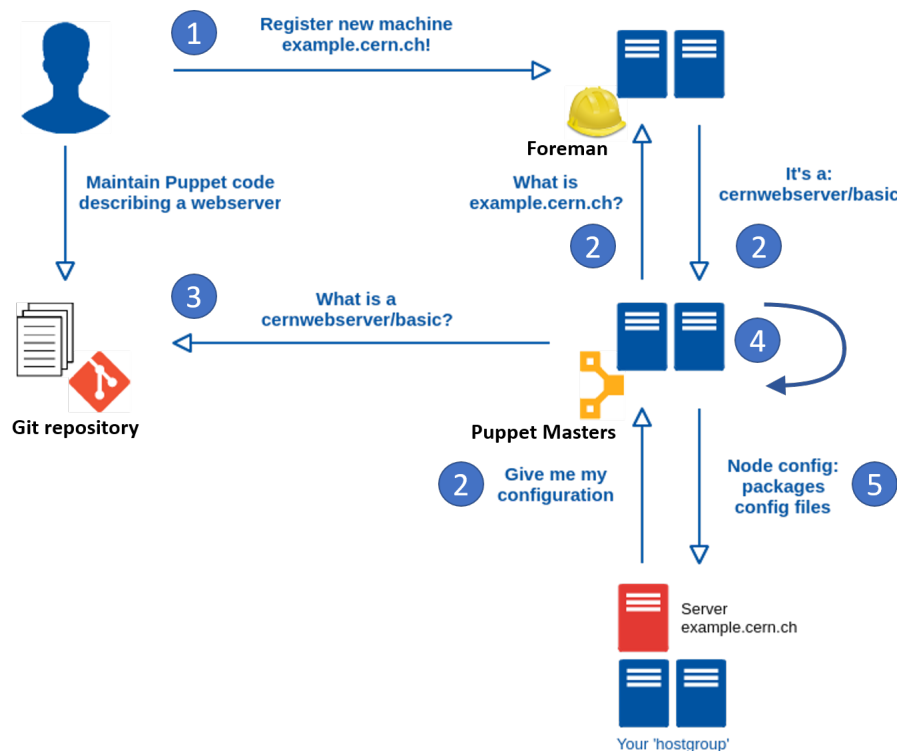


Figure 4.3: An overview of a puppet run - only steps 2 to 5 (adapted from [97]).

4. Puppet Master then compiles the desired configuration for that node and hands it back to the node's Puppet Agent for it to apply.
5. If the node has any configuration that is not aligned with the desired configuration, then the appropriate changes are made by the Puppet Agent. If the configuration is already as desired, then Puppet Agent does nothing.

Steps from 2 to 5 describe a puppet run (see fig. 4.3).

### Foreman Hostgroups, Git and Puppet at CERN

A *hostgroup* is Foreman's concept of a group of computing nodes to facilitate the configuration management of multiple hosts. By aggregating hosts in the same hostgroup implies that they are all part of the same service, have some configuration in common and are managed by the same group of people (SysAdmins).

Hostgroups are hierarchical which means that there is a top-level hostgroup representing the main service and there can be *sub-hostgroups* to represent different parts of the service.

Similarly to sub-class concepts (class inheritance), the configurations added to any hostgroup are inherited to the corresponding sub-hostgroups. For example, in the context of the project we are part of, we have a development hostgroup called *windows\_dev* to aggregate all the servers (Linux and Windows) part of the WTS infrastructure. Inside this

we have two sub-hostgroups, one for load balancers called *lb* and another for the Remote Desktop servers called *ts* (see table 4.1).

Table 4.1: Foreman Hostgroups.

Hostgroup	Purpose
windows_dev	The top-level hostgroup for the testing WTS infrastructure
windows_dev/lb	The hostgroup containing the load-balancer nodes of the testing WTS infrastructure
windows_dev/ts	The hostgroup containing the Remote Desktop servers of the testing WTS infrastructure

Foreman also allows to set a specific environment to a hostgroup or a individual host enabling SysAdmins to easily change which manifest (code) version should be applied to each machine. An environment describes which git repository branches to use to fetch the code, by default there are two: *qa* (Quality Assurance) and *production*. Each git repository that stores Puppet manifests has also two default branches: *qa* and *master*. Choosing the *qa* environment points the hosts to fetch code from the *qa* branch, whereas selecting *production* makes hosts fetch the code located in *master* branch.

All git repositories containing Puppet code follow a specific directory layout to ensure uniformity and to help the Puppet compiler to know where to look for the code. This layout is organised as follows:

- **Code** - where the Puppet code (manifests, templates and files) is stored.
  - Manifests (stored in *code/manifests*) - directory containing Puppet manifests.
  - Files (stored in *code/files*) - directory where raw configuration files or other kind of files should be stored so that they can be copied to the target nodes; manifests describe exactly where they get copied to on the target system. This directory is not mandatory, it should be used only when there's the need to copy such files.
  - Templates (stored in *code/templates*) - directory where template files are stored. Template files are files with placeholders that can be substituted by values provided in the manifest.
- **data** - where the Hiera data for the hostgroups is stored.
  - Hostgroup (stored in *data/hostgroup*) - contains the YAML files that provide key-value pairs for the hostgroup compilation. These values are used at compilation time.

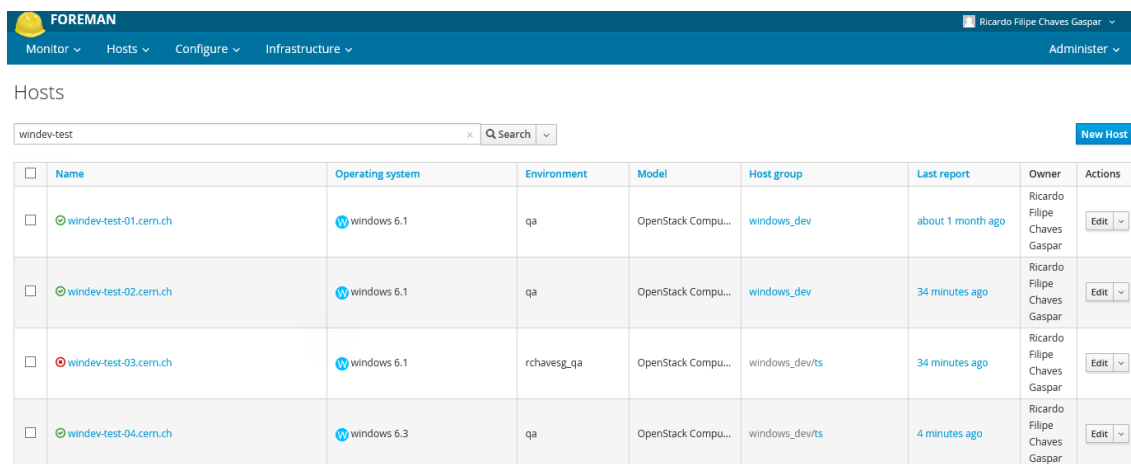
When looking for which manifests to compile, Puppet compiler considers the full hostgroup path, meaning, if it is a sub-hostgroup then it will traverse and compile from the top-level hostgroup manifest to the bottom (see table 4.2).

Table 4.2: Puppet code and hostgroup organisation.

Hostgroup element	File to look inside	Classname to use
windows_dev	<i>code/manifests/init.pp</i>	<i>hg_windows_dev</i>
windows_dev/lb	<i>code/manifests/lb.pp</i>	<i>hg_windows_dev::lb</i>
windows_dev/ts	<i>code/manifests/ts.pp</i>	<i>hg_windows_dev::ts</i>

Manifests directory represents the top-level hostgroup, it contains a top-level manifest called *init.pp* that is applicable to all the sub-hostgroups. For a host in the *ts* hostgroup, Puppet compiler will load and compile first the top-level manifest *init.pp* followed by *ts.pp*. If there were other sub-hostgroups, the compilation process would continue percolating down until the final sub-hostgroup that the host belongs to.

Figure 4.4 illustrates the Foreman web portal with some hosts - members of the *windows\_dev* hostgroup - and their information.



Name	Operating system	Environment	Model	Host group	Last report	Owner	Actions
<input type="checkbox"/> <a href="#">windev-test-01.cern.ch</a>	windows 6.1	qa	OpenStack Compu...	windows_dev	about 1 month ago	Ricardo Filipe Chaves Gaspar	<a href="#">Edit</a>
<input type="checkbox"/> <a href="#">windev-test-02.cern.ch</a>	windows 6.1	qa	OpenStack Compu...	windows_dev	34 minutes ago	Ricardo Filipe Chaves Gaspar	<a href="#">Edit</a>
<input type="checkbox"/> <a href="#">windev-test-03.cern.ch</a>	windows 6.1	rchavesg_qa	OpenStack Compu...	windows_dev/ts	34 minutes ago	Ricardo Filipe Chaves Gaspar	<a href="#">Edit</a>
<input type="checkbox"/> <a href="#">windev-test-04.cern.ch</a>	windows 6.3	qa	OpenStack Compu...	windows_dev/ts	4 minutes ago	Ricardo Filipe Chaves Gaspar	<a href="#">Edit</a>

Figure 4.4: Foreman Hostgroup example.

### 4.2.3 Ansible

#### Introduction

Ansible was first released by Michael DeHaan in 2012 as a small side project, and since then it gained a lot of traction and thousands of contributors on Github. As result of its popularity, RedHat acquired Ansible in 2015 and, despite having an Enterprise version, it still remains open source.

It is written entirely in Python and the main runner and all modules are compatible with Python 2.6, meaning that they work with any version of Python2 above version 2.6. DeHaan choose Python for Ansible as it would not require additional dependencies on the machines that needed to be managed [88].

Ansible follows a push-based model and does not require any agent on the target machines; it manages them by using remote management frameworks that already exist

natively on operating systems, like SSH for Linux/UNIX and WinRM for Windows [98]. Note that WinRM [99] enables the remote management of Windows machines as it is the Microsoft implementation of the WS-Man protocol [100] - a firewall-friendly protocol based on SOAP [101] that allows hardware and operating systems, from different vendors, to interoperate.

Furthermore, Ansible relies on the authentication methods of such frameworks and does not require any dedicated users or credentials. Hence, it does not require administrator access, leveraging *sudo*, *su*, and other privilege escalation methods on request when necessary.

Access to the control server (or source control) does not automatically grants access to remote hosts, as any user that wants to push content and/or exercise remote control must also have credentials on the remote systems.

Similarly, by following a push-based model where only needed code (called Ansible “modules”) is sent to remote machines, these cannot observe or interfere on how other machines are configured. Ansible also allows scripts and commands to be executed, thus offering some orchestration capabilities.

In addition, the agentless property means that no additional resources are consumed on managed machines when Ansible is not actively managing them. Altogether, these attributes are ideal for high-security environments or high-performance cases where there are concerns about the stability or existence of a local management agent, but are generally useful benefits in all computing areas [102].

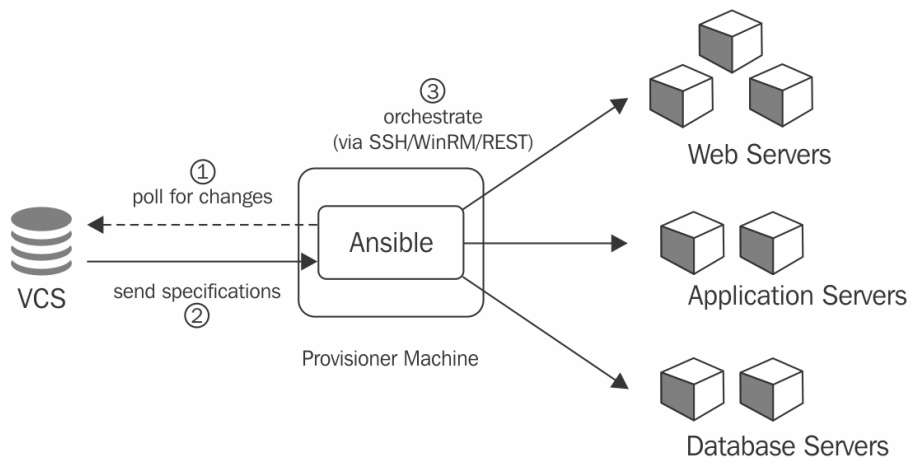


Figure 4.5: Ansible architecture overview [103].

Fig. 4.5 shows a general overview of how Ansible works. Similarly to the push-based model, the Ansible Control Machine (named Provisioner Machine in the figure) can retrieve the configuration files stored in a Version Control System (VCS) and apply them to the remote machines. The Ansible Control Machine has access to both the VCS and the target machines and has the ability to deliver specific configurations for different machine groups. This machine has to run Linux and acts like a master server, as it is

the fundamental entity that communicates with the remote machines, which may run Linux/UNIX or Windows.

In Ansible, configurations are idempotent and the configuration files, called playbooks, are written in a declarative paradigm using YAML - a data representation language (as JSON or XML) that intentionally tries to not be a programming language or script, but rather a model of a configuration or a process. YAML is commonly used for data configuration due to a simple syntax which makes it easy to read and use.

Besides playbooks, there are other elements that compose Ansible's architecture (see fig. 4.6):

- **Inventories** - files describing hosts and groups of hosts which need to be managed;
- **Modules** - programs provided by Ansible (Core Modules) and the open source community (Custom Modules) that enable the configuration and management of system resources (e.g. services, software packages, files). Modules are usually written in Python, but can also be written in PowerShell, and are directly executed on remote hosts through playbooks;
- **Plugins** - pieces of code that augment Ansible's core functionality. For example, Action plugins are like front-ends to modules and can execute tasks on the controller before calling the modules themselves;
- **APIs** - allowing interaction with, e.g., cloud services, public or private;
- **Ansible Config** - a file that describes the configuration settings of Ansible software installed in the Control Machine. For example, it describes where the module's library folder is located in the local filesystem.

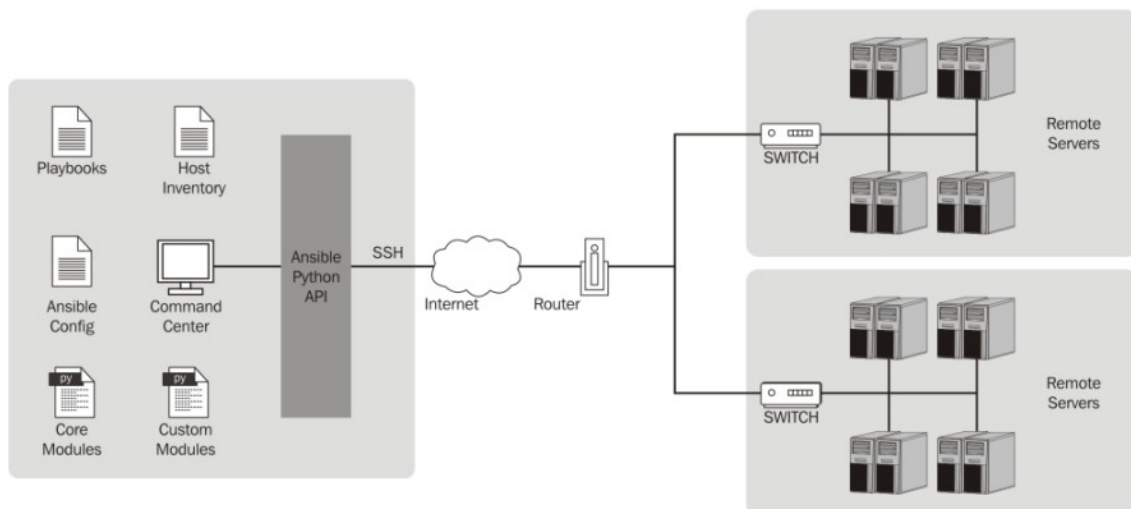


Figure 4.6: Ansible components [104].



### YAML Basics

YAML uses an approach based on indentation to describe data in a structured and hierarchic way. It relies on lists and dictionaries to structure data; listing 3 shows a few examples of the YAML syntax.

Although not mandatory, a YAML document usually starts with three dashes (“- - -”) and ends with three dots (“...”) to denote its beginning and end.

A list may be declared using a name followed by a colon and a set of additional lines, all beginning at the same indentation level, and starting with a dash and a space (“- ”) to describe the members of the list. A dictionary is represented in a simple *key: value* form where the colon must be followed by a space. In YAML, spacing is important because it defines the structure of data.

---

```
1  ---
2  # A list of fruits
3  fruits:
4      - Apple
5      - Orange
6      - Strawberry
7      - Mango
8
9  # An alternative syntax for a list
10 fruits: ['Apple', 'Orange', 'Strawberry', 'Mango']
11
12 # A dictionary representing an employee record
13 martin:
14     name: Martin D'vloper
15     job: Developer
16     skill: Elite
17
18 # An alternative syntax for a dictionary
19 martin: {name: Martin D'vloper, job: Developer, skill: Elite}
20
21 # A dictionary with a list value
22 - martin:
23     name: Martin D'vloper
24     job: Developer
25     skills:
26         - python
27         - perl
28         - pascal
29  ...
```

---

Listing 3: YAML syntax examples [105].

### Inventory and Playbooks

In Ansible, the way remote systems are configured and managed is fairly simple: one must define groups of hosts in the inventory file and describe the configurations in playbooks targeted at specific groups. Fig. 4.7 demonstrates how inventory and playbooks are used by Ansible to manage configurations in the remote machines: both are stored in the Ansible Control Machine (named “Management Node” in the figure) and, upon explicit invocation of the playbook, the configurations are applied to the target host

groups. A playbook file can have configurations for different host groups, so the same playbook can do different actions in each group of machines.

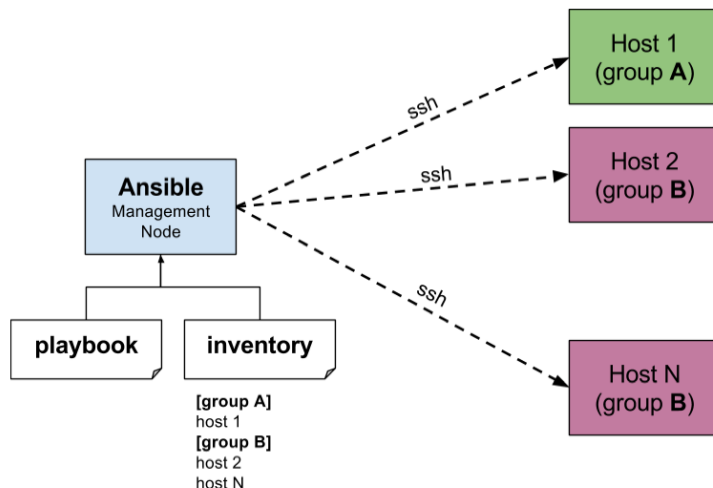


Figure 4.7: Inventory and playbooks in Ansible [106].

The inventory file, also called hosts file, can be written in many formats, but Ansible uses mainly two formats: INI (.ini) and YAML (.yaml or .yml). Listing 4 shows an example of an inventory file using the INI format: it contains one host (machine) without a group, and two host groups (*webservers* and *databases*), each with two machines. By default, all the hosts are members of the *all* group, so that playbooks can target all the machines described in the inventory. The YAML version of the example inventory is illustrated in listing 5.

---

```
1 mail.example.com
2
3 [webservers]
4 foo.example.com
5 bar.example.com
6
7 [dbservers]
8 one.example.com
9 two.example.com
```

---

Listing 4: Example of an inventory file using the INI format.

Playbooks are also expressed in YAML and they are composed by one or more “plays” in a list. The goal of a play is to map a group of hosts to some well defined roles, represented by what Ansible calls tasks, which essentially are no more than calls to module. Each task has a name and is followed by the module’s name and attributes. Tasks describe the resources to configure, manage or execute (services, file, software packages, etc.). Listing 6 shows an example of a playbook with just one play, while listing 7 depicts a playbook with two plays.

```
1 ---
2 all:
3   hosts:
4     mail.example.com
5   children:
6     webservers:
7       hosts:
8         foo.example.com:
9         bar.example.com:
10    dbservers:
11        hosts:
12          one.example.com:
13          two.example.com:
14 ...
```

---

Listing 5: Example of an inventory file using the YAML format.

```
1 ---
2 - hosts: webservers
3   vars:
4     http_port: 80
5     max_clients: 200
6     remote_user: root
7   tasks:
8     - name: ensure apache is at the latest version
9       yum: name=httpd state=latest
10    - name: write the apache config file
11      template: src=/srv/httpd.j2 dest=/etc/httpd.conf
12    - name: ensure apache is running (and enable it at boot)
13      service: name=httpd state=started enabled=yes
14 ...
```

---

Listing 6: Playbook with one play.

```
1 ---
2 # play 1
3 - hosts: webservers
4   remote_user: root
5   tasks:
6     - name: ensure apache is at the latest version
7       yum: name=httpd state=latest
8     - name: write the apache config file
9       template: src=/srv/httpd.j2 dest=/etc/httpd.conf
10
11 # play 2
12 - hosts: databases
13   remote_user: root
14   tasks:
15     - name: ensure postgresql is at the latest version
16       yum: name=postgresql state=latest
17     - name: ensure that postgresql is started
18       service: name=postgresql state=started
19 ...
```

---

Listing 7: Playbook with two plays.

#### 4.2.4 PowerShell DSC

##### Introduction

Windows PowerShell Desired State Configuration (DSC) is a configuration management platform based on open standards that allows SysAdmins to define the desired configuration state of a host and ensure that the machine is always kept in that state. The notion of desired state covers a lot, from installed software to configuration settings, from the OS to applications and services. DSC is designed to handle all this configuration data and execute it consistently and repeatedly; furthermore, DSC configurations are idempotent.

PowerShell DSC is a feature of PowerShell that is part of the Windows Management Framework (WMF) and requires PowerShell version 4 or later. Recently Microsoft has extended the support for Linux, releasing PowerShell for Linux [107].

DSC consists of three main components: configurations, resources and the Local Configuration Manager (LCM) [10].

- **Configurations** - scripts written in PowerShell Desired State Configuration DSL - a language with a PowerShell-based syntax and a declarative paradigm, used to define and configure instances of resources;
- **Resources** - code included in PowerShell modules that allows us to manage and configure system resources (files, services, server roles, OS settings, applications, etc.). A PowerShell module can provide multiple DSC resources;
- **Local Configuration Manager (LCM)** - the engine by which DSC facilitates the interaction between resources and configurations. The LCM regularly polls the system using the control flow implemented by resources to ensure that the state defined by a configuration is maintained. If there is a configuration drift, the LCM makes calls to the code in resources to re-apply the desired configuration.

DSC requires both a data file and a configuration file which are later translated into a text file that uses the MOF [108]. This file is then parsed and executed on the target machine, using DSC features that know how to configure the system.

The MOF was defined by the Distributed Management Task Force (DMTF), which is a vendor-neutral organisation that works towards standardised inter-operation between distinct platforms. The DMTF defined the MOF syntax and format so that any vendor or system can implement it, making it possible for third-party tools to manage configurations on Windows machines.

PowerShell DSC can be deployed using either a push or a pull model to manage configurations (see fig. 4.8). In either model, DSC uses an auxiliary host: a “central” workstation to push configurations to the target machines, in the push model, or a server (a.k.a. Pull Server) in the pull model. In the push model, the workstation can be a Windows computer with the required PowerShell and WMF versions in order to run the

DSC configuration scripts, whereas in the pull model, a pull server as to be set up to run the DSC Service and the clients (target nodes) must be configured to fetch configurations from the Pull Server.

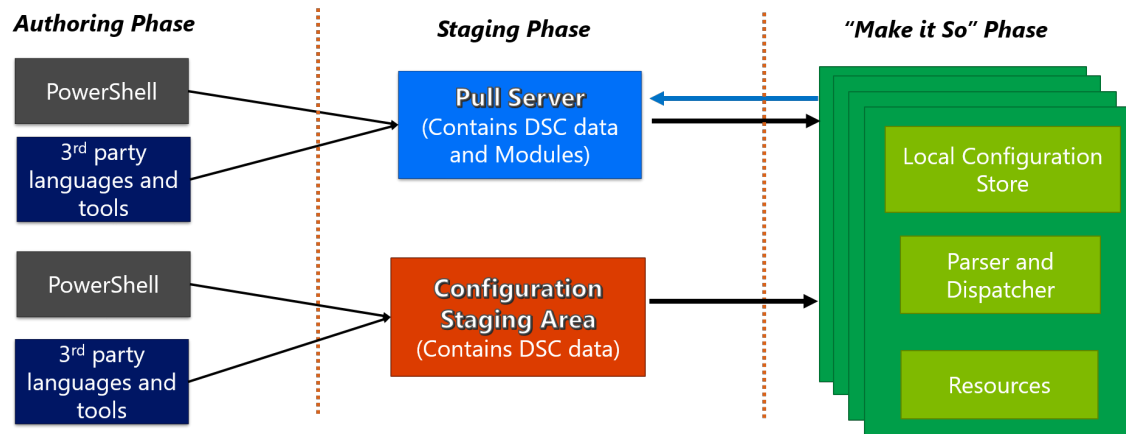


Figure 4.8: PowerShell DSC push and pull models overview (adapted from [109]).

Both approaches are composed by three phases [110]:

- **Authoring** - the DSC configuration is created through PowerShell or by third party languages and tools. The output from the Authoring Phase is one or more MOF files, the format which is consumable by DSC;
- **Staging** - DSC data (MOF files) is staged. In case of adopting the pull model, DSC data and custom providers are kept on the Pull Server; otherwise, when using the push model, DSC data is pushed to the target system;
- **Execution** (a.k.a. “Make it so”) - applies the MOF files that are either pulled or pushed to the “Local Configuration Store” (part of the LCM) that contains the current, previous and the desired state configuration. The configuration then gets parsed and the relevant resources implement the changes. When the LCM executes the MOF successfully, it renames the *pending.mof* file to *current.mof* file (see fig. 4.9).

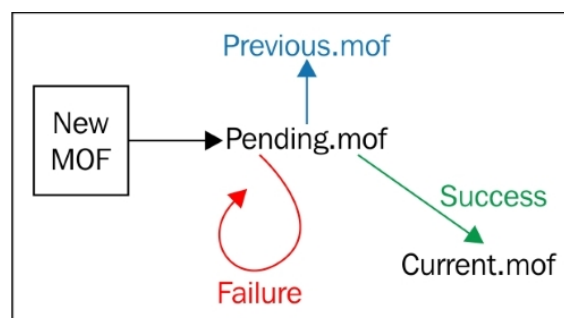


Figure 4.9: PowerShell DSC execution phase [111].

Figure 4.10 illustrates how the DSC push model works: first, the PowerShell configuration script (PS) has to be written, then compiled to a MOF file and finally executed by the target machine's LCM. In the push model, configurations must be manually pushed to the machines using a cmdlet (command line program) and the DSC resources must already be available in the machines; if not, they must be installed first.

This model also enables to push DSC configurations locally in a standalone manner, as they follow the same workflow. This is the easiest to set up and most flexible of the two DSC methods, but the hardest to maintain in large installations and in the long term.

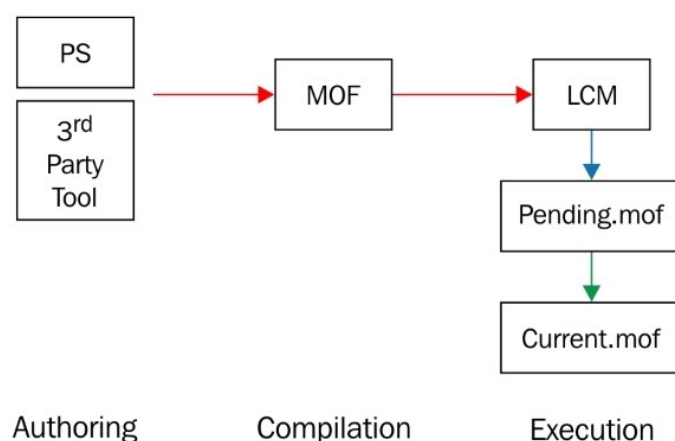


Figure 4.10: PowerShell DSC push model [111].

In a pull scenario, illustrated in fig. 4.11, the target machines have their LCM configured to periodically contact the Pull Server and fetch the latest configurations (MOF files) as well as the required DSC resources to execute such configurations. Each remote machine contacts the Pull Server using the server's Unique Resource Identifier (URI) and passes its name or a unique identifier to retrieve its specific configuration and verifies if all the custom resources are available; if not, those are downloaded to the target system.

Of the two DSC methods, the Pull Server is the harder to set up, but the easiest to maintain in large node installations and in the long term; it is suitable in server environments that have a lot of transient machines, like cloud or virtualised data centre environments where servers are created and destroyed frequently, and DSC configurations can be applied on a triggered basis.

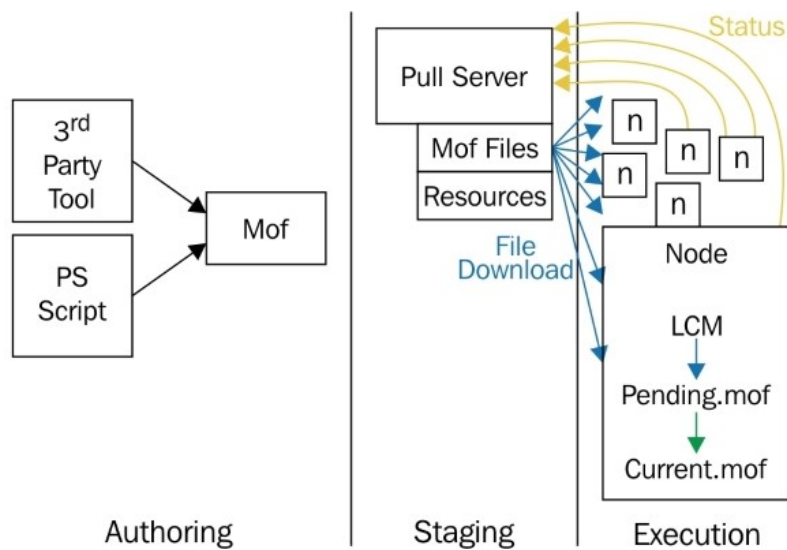


Figure 4.11: PowerShell DSC pull model [111].

### PowerShell DSC Scripts

PowerShell DSC configuration scripts are, in short, PowerShell scripts that define a special type of function that describes the configuration; they have the following structure:

- **Configuration** block - the outermost script block defined by the *Configuration* keyword, followed by the name of the function.
- **Node** blocks - one or more blocks inside the *Configuration* that define the configuration's target nodes (computers).
- **Resource** blocks - one or more blocks inside the *Node* block that describe the configurations and their properties. Resources are specified their names and followed by the name chosen for the configuration; settings are defined in the form *property = value*, using a declarative approach.

Only *resource* blocks are declarative, since in a *Configuration* block it is possible to write PowerShell (imperative) code. In addition, DSC configuration scripts are allowed to have PowerShell code before or after the DSC Configuration block, i.e., a DSC configuration script could be nested inside a larger script or in a standalone script. For that reason, they also have the same file extension (.ps1) as PowerShell scripts.

An example of a DSC configuration script is depicted in listing 8. It describes the installation of a web server with its first HTML file. The *Node* block specifies the target node to be configured, in this case the *localhost*. The configuration calls two resources: *WindowsFeature*, to install a web server (in this case, Microsoft IIS - Internet Information Server) and *File* to copy a HTML file to the web server's content folder. Resources ensure that the target node is/will be in the state defined by the configuration.

```
1 ---
2 Configuration WebsiteTest {
3
4     # Import the module that contains the resources we're using.
5     Import-DscResource -ModuleName PsDesiredStateConfiguration
6
7     # The Node statement specifies which targets this configuration will be applied to.
8     Node 'localhost' {
9
10         # The first resource block ensures that the Web-Server (IIS) feature is enabled.
11         WindowsFeature WebServer {
12             Ensure = "Present"
13             Name    = "Web-Server"
14         }
15
16         # The second resource block ensures that the website content copied to the
17         ↪ website root folder.
18         File WebsiteContent {
19             Ensure = 'Present'
20             SourcePath = 'c:\test\index.htm'
21             DestinationPath = 'c:\inetpub\wwwroot'
22         }
23     }
```

---

Listing 8: PowerShell DSC configuration script example [112].

## 4.3 SCM Tools' Evaluation

The SCM tools described in the previous sections (Puppet, Ansible and PowerShell DSC) enable the management of configurations of multiple systems in a programmatic way. Each tool has its own, particular, set of features and properties; therefore, in order to figure out which one would serve us better in the process of automating the configuration of the WTS infrastructure, we had not only to compare their features, but also to effectively use them on the field.

We've started the process by assembling, in table 4.3, the most relevant characteristics, in order to summarise and better compare them against each other.

### 4.3.1 Tool Installation and Configuration

#### Puppet

In case of Puppet Master no deployment was needed, as we used CERN's already in-place Puppet installation. Anyway, it is important to stress that, in terms of the installation procedure, Puppet is the most difficult tool to deploy as it requires more configuration steps, and the target nodes must have the agent installed and configured to contact the Puppet master. Puppet also requires the installation of SSL certificates on both parties: for an agent to communicate with the master it must generate its own certificate, based on the master's certificate, and then send it to the master to have it signed by the master.



Table 4.3: Characteristics of the evaluated SCM tools.

	<b>Puppet</b>	<b>Ansible</b>	<b>PowerShell DSC</b>
<b>Architecture</b>	Master-Agent	Agentless	Master-Agent
<b>Delivery model</b>	pull	push	push/pull
<b>Configuration program terminology</b>	manifest	playbook	DSC configuration script
<b>Configuration Language</b>	Puppet DSL	YAML	PowerShell DSC DSL
<b>Scalability (manage multiple nodes)</b>	Yes	Yes	Yes
<b>Basic resources (create files, install packages, manage services)</b>	Yes	Yes	Yes
<b>Support for Custom/Community Modules</b>	Yes	Yes	Yes
<b>Module Repositories (Official and Community)</b>	Puppet Forge and GitHub	Ansible Galaxy and GitHub	PowerShell Gallery and Github
<b>Management, Monitoring and Reporting Visualisation solutions</b>	Free/Open Source: The Foreman [113] and Puppet Dashboard [114]	Free/Open Source: The Foreman, ARA [115] and Tensor [116]	Non Existent, CLI or Event Viewer (Windows default logging visualisation program)
	Commercial: Puppet Console (part of Puppet Enterprise) [117]	Commercial: Red Hat Ansible Tower [118]	
<b>SCM tool platform</b>	Masters must run on Linux	Ansible Control Machine must run on Linux	Pull Server must run on Windows Server 2012 or higher
<b>Target platform</b>	Windows/Linux	Windows/Linux	Windows/Linux
<b>Windows-specific resources</b>	Yes	Yes	Yes
<b>Orchestration capabilities</b>	Yes	Yes, built-in	Not Applicable
<b>Redundancy</b>	Yes	No	No

As referred above, as we used CERN's Puppet infrastructure, we didn't have to face the challenge of installing and and configuring Puppet Master(s). The only step needed was to install Puppet Agent on the target machines, which was done by deploying a pre-configured software package using CERN's CMF tool (see section 2.3.4).

## Ansible

Ansible was easy: for the Ansible Control Machine, we set up a CentOS 7 VM in CERN's OpenStack using a *m2.small* flavour; installation was just the download/install of the software package (using the OS package manager - *yum*); and configuration was accomplished through modifications of the configuration file. Although Ansible doesn't need agents on the managed hosts, Windows systems need to have WinRM properly configured. But, thankfully, on the Ansible website there is a PowerShell script that configures the WinRM service; so all we had to do was run the script on each machine.

## PowerShell DSC

Setting up a PowerShell DSC Pull Server was a bit more difficult than setting Ansible, but thanks to the DSC configuration scripts made available by Microsoft the process was not very difficult (note that DSC configuration scripts can be always be executed locally, in a standalone approach, when using the push model). The Pull Server was an OpenStack-based VM with a *m2.large* flavour, running Windows.

The installation of the Pull Server required the generation of a Global Unique Identifier (GUID) to serve as registration key so that clients could register themselves against the server. This registration key is considered a secret that should only be shared between the server and the clients. In the target machines a DSC configuration script as also executed to set them up, which included the Pull Server's URI and the registration key.

### 4.3.2 Testing the Tools

These tools were tested in a practical scenario with two Windows machines, posing as clients (configuration targets), each running a different Windows version: Windows Server 2012 R2 and Windows Server 2016. The goal was to use them to a) evaluate each tool's demand on the clients, i.e., what to install and configure, and b) see the tool at work. The clients were, in fact, *m2.large* VMs (see flavours in 2.1.1) running on CERN's OpenStack private cloud service.

Each tool was used to configure the two target server machines and, after finishing the tests with one tool the machines were recreated to ensure that they were clean and ready to be configured again.

To test and compare the tools, we created a few configuration programs in each tool's language to test some standard resources (available in all languages), such as copying files and manage services (see annexes I,II,III). All tools evaluated here do apply configurations, ensure idempotency and fix configuration drifts essentially the same way; the differences reside in the programming language.

Nevertheless, DSC lacks the flexibility of the remaining tools as it does not offer a centralised way of managing the host groups. In DSC, configurations (MOF files) have a name (the name of the configuration file) and are stored in the Pull Server; each machine must be configured to track a specific configuration (using the configuration name) and, therefore, to use a different configuration for a machine, the machine LCM must

be reconfigured to track the new configuration. If the configuration is updated (while keeping the same name), machines are still able to fetch and apply the new configuration. This is an important aspect as it makes the management of configurations and machines harder. Puppet and Ansible use hostgroups or an inventory file, something that allows us to centrally manage groups of machines and the configurations targeted to those groups.

### 4.3.3 Integration with PowerShell DSC

#### Introduction

As PowerShell DSC allows to configure and manage some resources (e.g. manage Group Policy rules) that are not available in the other SCM tools, but lacks some of the capabilities and features they possess (like management of hosts groups), we decided to evaluate the integration of Puppet and Ansible with PowerShell DSC.

Since PowerShell DSC allows other tools to use DSC resources, they can easily extend the set of resources they manage by providing integration modules that bridge this gap. Puppet and Ansible have such modules, and they enable the use of DSC resources in their languages by offering the same syntax as their standard resources.

Puppet uses the *puppetlabs-dsc* to use DSC resources, whereas Ansible uses *win\_dsc* module. Despite having the same purpose, these modules work differently, as Puppet and Ansible have different architectures.

#### Puppet integration

In Puppet, the Puppet Master has all the “native” modules (bundles of resources) and only sends them to the agent when they are present in the manifest; the agent then saves the resources (Ruby code) locally for future runs (executions).

For the DSC module, Puppet does the same thing: only DSC resources in the manifest are sent to the agent. One important thing to note is that DSC resources are written in PowerShell code, but the *puppetlabs-dsc* module already comes with plenty of them compiled to Ruby, so they can be used in manifests as “native” Puppet resources (a.k.a. types). The module also provides a script that allows importing and compiling custom DSC resources made available by the PowerShell community.

#### Ansible integration

Since Ansible is agentless, it takes a simpler approach that does not require the Control Machine to transfer the DSC modules to the managed nodes. Instead, the *win\_dsc* assumes that the modules are already installed on the machines and remotely invokes (through WinRM) the DSC resources. Hence, to use DSC resources, they must first be installed on the remote machines.

#### Integration test

The goal of the test was to compare Puppet and Ansible approaches in terms of ease of use and check if they apply the resources in the same way that PowerShell DSC does. The

test, in itself, was: "Force the Windows system to use a specified server as its Windows Licensing Server"; in the test mockup, the server's FQDN is *server.test.localgpo.dsc.com*. To have a visual idea of how a SysAdmin could set it manually, using the Local Group Policy Editor GUI (see fig. 4.12): he/she would first navigate to the Licensing Folder, then Select the rule, and then change the setting to Enabled and fill the "License servers to use" field with the appropriate FQDN (see fig. 4.13).

We have developed three different configuration programs, one for each tool; each program uses the same community module, available on PowerShell Gallery, called *PolicyFileEditor*. The module allows us to set Group Policy rules on Windows hosts and contains a resource called *cAdministrativeTemplateSetting* that allows us to set a Group Policy rule called Administrative templates (see section 2.3.2) which, finally, allows the desired goal to be reached.

As a baseline, we developed a PowerShell DSC configuration script that sets the above-mentioned rule; a sample of the code is depicted in listing 9, while the full code is available in annex IV.

To apply the DSC configurations, the nodes' LCM was configured to fetch the DSC configuration (MOF file) from the Pull Server; this initial step, executed on each machine's LCM, also registers the host against the Pull Server. The file contents were created using the DSC configuration script as its basis.

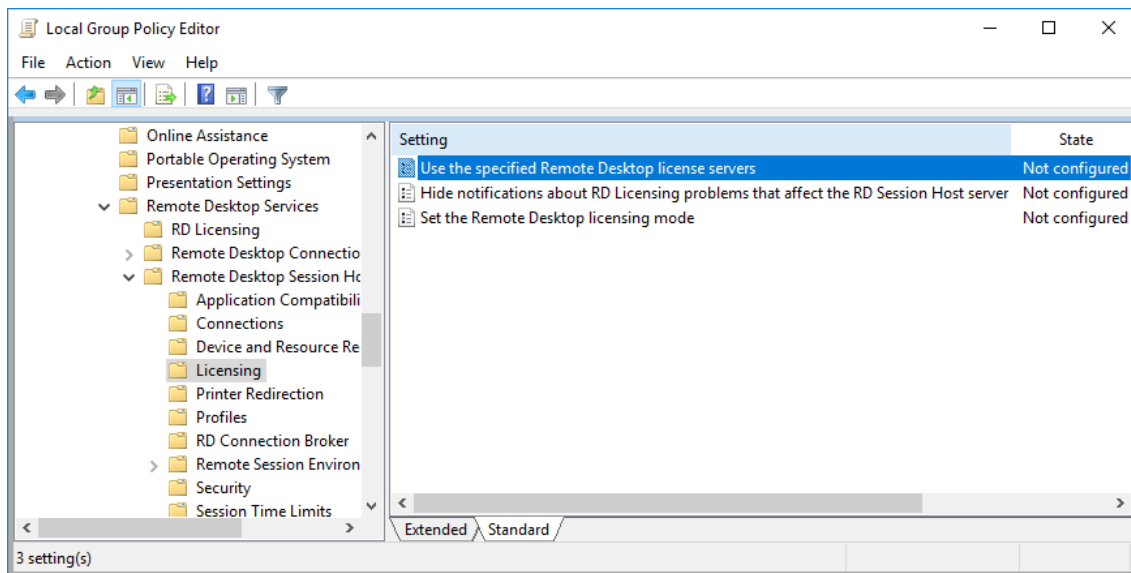


Figure 4.12: Group Policy GUI displaying the rule to set.

### Puppet Integration test

For Puppet, we had to use the *puppetlabs-dsc* module to generate the corresponding Puppet resources (in Ruby) from the *PolicyFileEditor* DSC module. Once compiled, the resources had to be uploaded to the *puppetlabs-dsc* module repository at CERN so that the Puppet Masters could find the resource when compiling the catalog. Only then, the

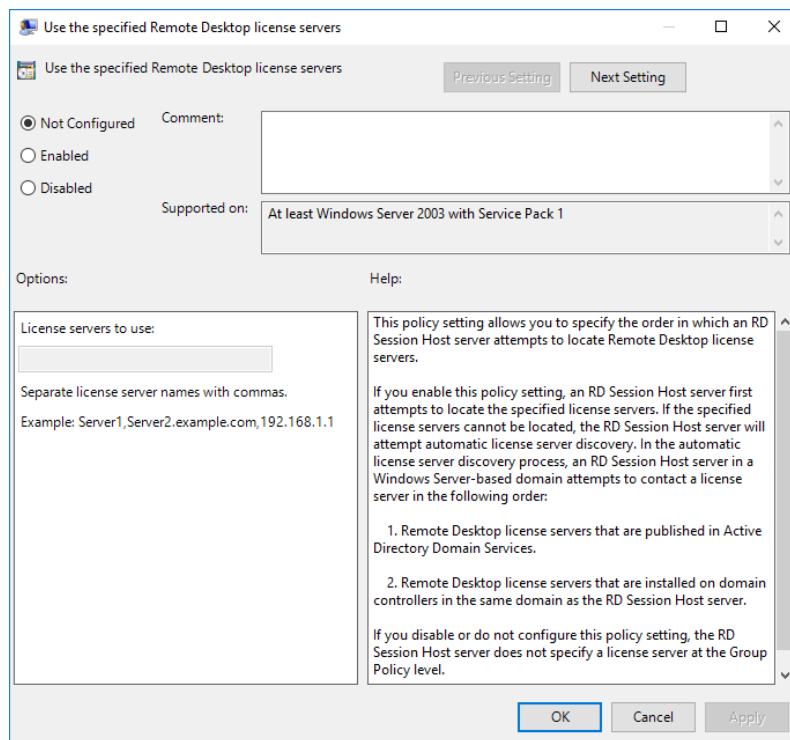


Figure 4.13: Group Policy GUI displaying the settings of the rule to set.

```

1 Configuration LocalGPO
2 {
3     param([string[]] $NodeName = 'localhost')
4
5     Import-DSCResource -ModuleName PolicyFileEditor
6
7     Node $NodeName
8     {
9         cAdministrativeTemplateSetting RDPLicensing
10        {
11            KeyValueName = "SOFTWARE\Policies\Microsoft\Windows NT\Terminal
12            ↪ Services\LicenseServers"
13            PolicyType = "Machine"
14            Data = ("server.test.localgpo.dsc.com")
15            Ensure = "Present"
16            Type = "String"
17        }
18    }
19 }

```

Listing 9: PowerShell DSC configuration script to set a Group Policy rule.

Puppet manifest equivalent to the DSC configuration script was developed and later applied to the target machines. A more detailed explanation, about this module and the work we have done with it, is available in subsection 4.4.5.

A sample of this code is depicted in listing 10 and the full code with the remaining Group Policy rules can be found in annex V. The main syntactic difference between

Puppet’s code and the PowerShell DSC configuration script in listing 9 is that both resource and attribute’s names are prefixed by “dsc\_” and are all lowercase. This is to avoid collisions with other Puppet resources and attributes.

To apply the configurations to the machines, the machines were added to a new hostgroup (in Foreman) and this manifest was assigned to the group. Later, Puppet agents could fetch and apply the correct configurations targeted at those machines.

---

```
1 class hg_windows_dev::dsc_tests {
2   # Remote Desktop Services\Remote Desktop Session Host\Licensing\Use the specified
3   dsc_cadministrativetemplatesetting { 'test-dsc_cAdministrativeTemplateSetting License
    ↪ Server':
4     dsc_keyvaluename => 'SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↪ Services\\LicenseServers',
5     dsc_policytype   => 'Machine',
6     dsc_data         => 'server.test.localgpo.dsc.com',
7     dsc_ensure       => 'Present',
8     dsc_type         => 'String',
9   }
10 }
```

---

Listing 10: Puppet manifest using a DSC to set a Group Policy rule.

### Ansible Integration test

The Ansible integration with DSC was simpler than the other tools’ approaches in some aspects. First, the playbook was simpler and easier to develop and read. Unlike Puppet, Ansible uses the original DSC resource and attribute names making it easier to transcode the PowerShell DSC script to YAML. Then, the assignment of hosts to a hostgroup was as simple as adding their names to the inventory file and define the group’s name (*windows-ansible* in this case). Finally, in addition to the WinRM setup already discussed in the previous section, the *PolicyFileEditor* module had to be installed on each machine so that the playbook could call its resources.

A sample of the playbook we developed is depicted in listing 11 and the full code is available in annex VI.

#### 4.3.4 Evaluation Results

The tests we have performed, and in particular the integration with PowerShell DSC, gave us a practical sense how these tools work and showed their advantages and limitations.

Puppet is the most mature of the tools tested as it keeps maintaining compliance across the managed nodes due to its master-agent architecture and pull model: agents periodically check with the masters if their current configuration state is up-to-date and, if not, the most recent one is pulled and applied. Another advantage is that it allows a simplified management of groups of machines and their configurations, and also provides detailed reports.

---

```

1  ---
2  - hosts: windows-ansible
3    tasks:
4      # GPO rules
5      # Remote Desktop Services\Remote Desktop Session Host\Licensing\Use the specified
6      - name: Set RDP Licensing server
7        win_dsc:
8          resource_name: cAdministrativeTemplateSetting
9          Ensure: "Present"
10         KeyValueName: "SOFTWARE\Policies\Microsoft\Windows NT\Terminal
           ↳ Services\LicenseServers"
11         PolicyType: "Machine"
12         Data: "server.test.localgpo.dsc.com"
13  ...

```

---

Listing 11: Ansible playbook using a DSC to set a Group Policy rule.

Puppet's drawbacks are: a complex installation process; the language's learning curve; the procedure required to integrate custom DSC resources; and, finally, it requires an additional product, MCollective, if one wants to centrally trigger the execution of manifests.

Ansible's agentless architecture and push model warrants a simple setup procedure. In addition, the language is very easy to learn, the management of groups of machines and their configurations is also simple, and the level of reporting is also very detailed (offering multiple verbose levels).

Ansible's drawbacks are: since it uses a push model, maintaining compliance requires a schedule task has to be set up in the Control Machine to periodically trigger the execution of a playbook. Therefore it does not scale well for environments where compliance is important. It follows a "set it up once and let it be" philosophy that gives SysAdmins more flexibility by allowing them to push configurations whenever they need.

It is fairly simple to integrate PowerShell DSC into Ansible, as it keeps the same names; however, PowerShell DSC modules have to be installed on each machine first, which requires another deployment step. Ansible scales well but, when compared to Puppet, requires more interactions between the managing and managed hosts.

PowerShell DSC is a tool that offers plenty of resources (both from Microsoft and the community) designed to configure and manage Windows OS as well as other Microsoft products. It benefits from the use of the PowerShell language and engine, which make it easy to learn and use - in particular for Windows SysAdmins that already develop PowerShell scripts. It provides the flexibility to use both push and pull models, but the management of groups and configurations in the pull scenario is more complex when compared to the other tools.

Ultimately, PowerShell DSC is not as mature as other SCM tools in terms of features; Microsoft has stated that the product is there to offer a programmatic way to configure their systems; therefore they welcome its integration with other, more powerful, SCM tools.

Table 4.4 reflects the conclusions drawn from the use and tests performed with these tools.

Table 4.4: Evaluation results for the surveyed SCM tools.

	<b>Puppet</b>	<b>Ansible</b>	<b>PoweShell DSC</b>
<b>Language Learning curve</b> (Easy, Medium, Hard)	Medium	Easy	Easy
<b>Infrastructure Set up</b> (Easy, Medium, Hard)	Hard	Easy	Medium
<b>Reporting and Logging (Debug)</b>	Detailed	Very Detailed	Detailed
<b>Integration with DSC</b> (use of custom resources)	Good but Difficult	Good and Easy	Not Applicable
<b>Ensure configuration compliance</b>	Good	Good	Good
<b>Configuration Management of groups of machines</b> (Easy, Medium, Hard)	Easy	Easy	Hard

In the end, Puppet was chosen to configure WTS infrastructure mainly because of three reasons: there is a dedicated team for the configuration management infrastructure at CERN that gives support to other users (i.e., other teams in the IT Department); using a different SCM tool would require a) the deployment of its infrastructure and b) the creation of another team; and, finally, the WTS's team limited human resources (only 3 people).

Furthermore, the adoption of Puppet was a decision from the IT department management board, and other teams should follow its strategy and adopt the chosen tools; however teams can come up with different tools if they find they solve the problems in a better way.

Although this evaluation didn't change the SCM tool adopted for the WTS infrastructure, it certainly showed what other tools have to offer.

## 4.4 WTS Configuration Management using Puppet

In the previous section, we have justified Puppet's choice as the SCM tool to configure WTS infrastructure. We will now see how Puppet allows the small team responsible for this infrastructure to deploy and manage configurations of its multiple hosts. As the focus is on the Remote Desktop hosts, the configurations we have developed are targeted to Windows Server systems running RDS.

Our contribution to a higher-level of automation in the management of WTS configurations, using Puppet, can be illustrated by manifest (script) depicted in annex XVIII; the work includes not only the re-use of standard Puppet but also community modules, **and our own contributions**, that we have made available to the community. As such,



we will now describe the Puppet modules that we have developed and/or modified and published.

##### 4.4.1 *Puppet-wmi* module

*puppet-wmi* was the first module that we have modified; in particular, we improved the configuration of a RDS through a specific property that could be changed using WMI. This module is a community module available on the Puppet Forge - the website with all community modules - and its source code could be found on GitHub repository; it defines a Puppet type (resource) that is able to change WMI settings - settings that can only be changed through WMI - using Puppet *exec* resources and PowerShell commands. The problem was that the module was not updated since 2014 and could only be used in specific cases - i.e., it was not sufficiently generic.

To understand the problem and how we addressed it, some context is necessary: WMI has classes to enable the manipulation of certain internal Windows settings and most, **but not all**, classes have methods (getters and setters) that return or change properties (variables). The original module (depicted in listing 12) was assuming that all properties had a corresponding *set* method, which is not true - an example being the RDS property we wanted to configure.

Consequently, the module needed to be extended in order to provide another way of changing a WMI setting - by changing the property directly without using a class method. Thus, we extended the module by changing the arguments' values, removing the *set* method assumption and writing the necessary code to solve the above scenario. Listing 13 shows the improved version. To browse the full code see annex VIII.

As a result of this improvement, the module can now be used by Puppet developers to apply similar configurations. The updated version of this module can be found in a repository available on GitHub [119] and also on Puppet Forge [120].

Note: for unknown reasons the author of the original module removed the module from the Git repository: at the time of this writing, only the module's description was available on Puppet Forge [121] but the detailed code can be found in annex VII.

##### 4.4.2 *puppet-sslcertificate* module

The automation of the installation of SSL certificates was very important in WTS; we found that there was a *puppet-sslcertificate* [122, 123] available specifically for that purpose on the Puppet Forge repository. This module defines a Puppet resource with the name *sslcertificate* that checks if a certificate is already present and, if not, installs it storing it in the Windows local certificate store.

To perform that task, it dynamically creates two PowerShell scripts and executes them. A generic version of the PowerShell scripts is stored in a template format (ERB - Embedded Ruby) and has template variables that Puppet replaces with variables declared

```
1 # == Defined Type: wmi
2 # This module is a defined type for manipulating WMI Objects with Puppet.
3 #
4 # === Examples
5 # wmi { 'Remote Desktop - Network Level Authentication' :
6 #   wmi_namespace => 'root/cimv2/terminalservices',
7 #   wmi_class      => 'Win32_TSGeneralSetting',
8 #   wmi_property   => 'UserAuthenticationRequired',
9 #   wmi_value      => 1,
10 # }
11 define wmi (
12   $wmi_namespace, $wmi_property, $wmi_class, $wmi_value, $wmi_method =
13     => "Set${wmi_property}") {
14
15   $wmi_array = ["-Namespace ${wmi_namespace}", "-Class ${wmi_class}",]
16   $wmi_data  = join($wmi_array, ' ')
17   $wmi_ps    = "Get-WmiObject ${wmi_data}"
18   $wmi_chk   = "If ((\${wmiobject}.${wmi_property}) -like '${wmi_value}')"
19
20   exec { $name :
21     command => "\${wmiobject}=${wmi_ps};\${wmiobject}.${wmi_method}(${wmi_value})",
22     onlyif  => "\${wmiobject}=${wmi_ps};${wmi_chk} { exit 1 }",
23     provider => powershell,
24   }
25 }
```

---

Listing 12: Original version of *puppet-wmi* module.

in the module's manifest by using the *template* function. The PowerShell (PS) code is, naturally, driven by the arguments passed to the resource. One script is responsible for verifying if the certificate is already installed, while the other one installs the certificate.

The existing version of the module was creating and storing the PS scripts in a hard-coded temporary location (in the target's machine filesystem) and was installing the certificate as exportable (default setting, if not explicitly set). The certificate's exportability property denotes if a certificate, after being installed, can be exported to other computers.

For WTS configuration, we needed this module but the above mentioned limitations were unacceptable due to security reasons (the VMs prepared by the WTS team for dedicated clusters are delivered to other departments/groups at CERN which can then administer the hosts, and thus have local access to the scripts and certificates).

So we went on and improved the module's code and script templates, as follows: in the code, two more arguments were added - one (named *scripts\_dir*) to specify the scripts' location, and another (named *exportable*) to set the exportability of the certificates (see listing 14). In addition, a conditional statement was added to pass the exportability argument to the script templates (see annex chapters XIII and XIV).

For comparison purposes, the original code and templates are listed in annex chapters IX to XI. As this is an open source module, these improvements were submitted and accepted in the GitHub repository [124].

```
1 # == Defined Type: wmi
2 # This module is a defined type for manipulating WMI Objects with Puppet.
3 #
4 # This example use specified methods.
5 # wmi { 'Remote Desktop - Allow Connections' :
6 #   wmi_namespace => 'root/cimv2/terminalservices',
7 #   wmi_class      => 'Win32_TerminalServiceSetting',
8 #   wmi_property   => 'AllowTSConnections',
9 #   wmi_value      => '1',
10 #   wmi_method     => 'SetAllowTSConnections',
11 # }
12 define wmi ($wmi_namespace, $wmi_property, $wmi_class, $wmi_value, $wmi_method = "") {
13   $wmi_array = ["-Namespace ${wmi_namespace}", "-Class ${wmi_class}",]
14   $wmi_data = join($wmi_array, ' ')
15   $wmi_ps = "Get-WmiObject ${wmi_data}"
16   $wmi_chk = "If ((\${wmiobject}.${wmi_property}) -like '${wmi_value}')"
17
18   if $wmi_method != "" {
19     $wmi_pscommand = "\${wmiobject}=${wmi_ps};\${wmiobject}.${wmi_method}('${wmi_value}')"
20   } else {
21     $wmi_pscommand = "${wmi_ps} | Set-WmiInstance -Arguments
22     ↪ @({'${wmi_property}'='${wmi_value}'}"
23   }
24
25   exec { $name:
26     command => "${wmi_pscommand}",
27     onlyif  => "\${wmiobject}=${wmi_ps};${wmi_chk} {exit 1} else {exit 0}",
28     provider => powershell,
29   }
```

---

Listing 13: Improved version of *puppet-wmi* module [119].

### 4.4.3 *cernsslcertificate* module

Although we were successful in extending the *puppet-sslcertificate* module to better suit our needs, it still had one problem: the certificate's password had to be passed as argument in plain-text and, consequently, the scripts stored in the target machines also had a plain-text password. Furthermore, in order to use this resource in the WTS puppet manifest, the password had to be hard-coded, and that is not a good practice for security reasons; Git repositories at CERN (GitLab) are shared between multiple users, and if by chance this code is copied or published online it can cause a security breach.

We found that the most elegant solution would be to create another module that served as wrapper around *puppet-sslcertificate* to a) take advantage of the improvements previously made and b) override some “standard” resources in order to adapt to the CERN's use case. As a consequence, *puppet-sslcertificate* could be decoupled and thus evolve/be updated independently, and the new module, called *cernsslcertificate*, would still be able to use the standard *puppet-sslcertificate*.

*cernsslcertificate* now installs the certificates and saves the scripts with the plain-text password in a more protected directory, only accessible by the SYSTEM account of the

```
1 define sslcertificate ($password, $location, $thumbprint, $root_store = 'LocalMachine',
2   $store_dir = 'My', $scripts_dir = 'C:\temp', $exportable = true) {
3
4   validate_re($name, '^(.)+$', "Must pass name to ${module_name}[${title}]")
5   validate_re($location, '^(.)+$', "Must pass location to ${module_name}[${title}]")
6   validate_re($thumbprint, '^(.)+$', "Must pass a certificate thumbprint to
   ↪  ${module_name}[${title}]")
7
8   ensure_resource('file', $scripts_dir, {
9     ensure => directory
10  })
11
12  if $exportable {
13    $key_storage_flags = 'Exportable,PersistKeySet'
14  } else {
15    $key_storage_flags = 'PersistKeySet'
16  }
17
18  file { "inspect-${name}-certificate.ps1":
19    ensure => present,
20    path   => "${scripts_dir}\\inspect-${name}.ps1",
21    content => template('sslcertificate/inspect.ps1.erb'),
22    require => File[$scripts_dir],
23  }
24
25  file { "import-${name}-certificate.ps1":
26    ensure => present,
27    path   => "${scripts_dir}\\import-${name}.ps1",
28    content => template('sslcertificate/import.ps1.erb'),
29    require => File[$scripts_dir],
30  }
31
32  exec { "Install-${name}-SSLCert":
33    provider => powershell,
34    command  => "${scripts_dir}\\import-${name}.ps1",
35    onlyif   => "${scripts_dir}\\inspect-${name}.ps1",
36    logoutput => true,
37    require  => [File["inspect-${name}-certificate.ps1"],
   ↪  File["import-${name}-certificate.ps1"]],
38  }
39 }
```

---

Listing 14: Improved version of *puppet-sslcertificate* module manifest.

Windows machine. Thanks to the usage of resources from another module called *teigi*, it was possible to still use the *sslcertificate* resource without having the plain-text password hard-coded in the manifest (see subsection 4.4.4).

The full code is depicted in annex XV and it shows the use of the *teigi* module and the overridden resources. Basically, *cernsslcertificate* calls *sslcertificate* resource providing a *teigi* key as password (see subsection 4.4.4) while overriding the existing *file* and *exec* resources (see listing 14) to solve the hard-coded password problem.

Hence, the strategy was to rename the scripts produced by the *sslcertificate*, using

with the “.withoutteigi” suffix, since they contain the *teigi* key not the certificate’s password. Figure 4.14 illustrates in a flowchart format the strategy while the next paragraphs describe it.

First the scripts are renamed, and then they are fed to the *teigi* resources to generate the final script files with the *teigi* key replaced by the certificate’s password. Furthermore, the *exec* resource that runs the scripts was also overridden so that it could execute only after the final scripts were produced by the *teigi::secret::subfile* resources.

Another important aspect is how these resources were overridden in order to not totally disrupt the resources declared in *puppet-sslcertificate*, but instead “inject” the *teigi* code in between the template resolution (replacement of variables) and the execution of the final scripts. The flowchart of figure 4.15 illustrates the execution steps of the *cernsslcertificate* module.

This overriding of resources was done using two features of the Puppet language: relationship attributes and resource collectors (a.k.a. spaceship operator).

As explained in the examples of subsection 4.2.2, relationship attributes allow to specify when the a resource should run; in this module, *before* and *require* attributes were used to force each *teigi::secret::subfile* to execute before the *exec* resource that would execute the scripts and after the corresponding *file* resources that generate the filled template scripts (without password).

The second feature, resource collectors, enable to select a group of resources by searching the attributes of every resource in the catalog. This search is independent of evaluation-order (that is, it even includes resources which haven’t yet been declared at the time the collector is written) [125]. The way to use it is through the spaceship operator: *ResourceType* <| *condition* |>. A usage example of spaceship operator can be found in a sample of the *cernsslcertificate* code (depicted in listing 15) where *file* resources are being collected based on their title (which have the same as each script name - stored in variables). When collecting the resource, its properties can be overridden. In the example, *path* attribute’s value is being overridden by another value - in this case the path is being set to the *scripts\_dir* directory’s path and the scripts are being renamed by adding the “.withoutteigi” at the end.

#### 4.4.4 *teigi* module and *teigi\_subfile* resource

The *teigi* module was developed in 2013 at CERN and its goal is to use secrets (e.g. passwords) on Puppet managed machines because the traditional Puppet mechanisms weren’t particularly helpful, since Puppet compiles the catalog on the Puppet Master.

*teigi* is not just a module, it has also a service infrastructure behind that offers a centrally protected key-value store for secrets. It uses the the node credentials (Kerberos) and the hostgroup membership to authorise the access to the secrets [126]. *teigi* allows to avoid hard-code the passwords by specifying a key (a.k.a. *teigi\_key*) that has a corresponding value (the real password) stored in the *teigi* centrally protected key-value store.

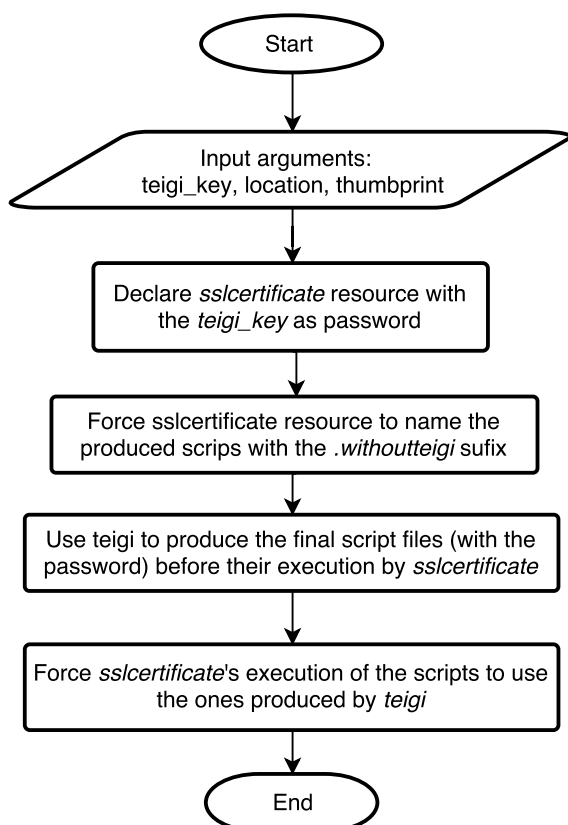


Figure 4.14: Cernsslcertificate logic flowchart.

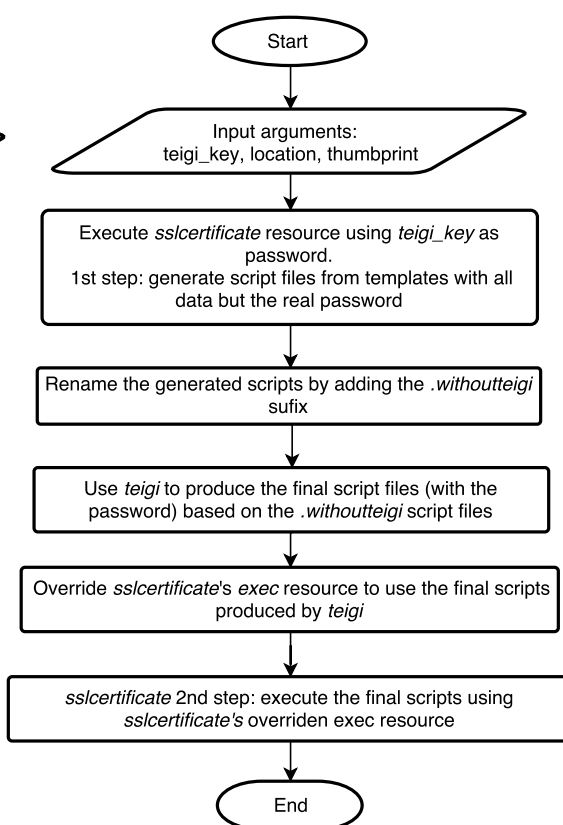


Figure 4.15: Cernsslcertificate execution flowchart.

```

1  $inspect_script_name = "inspect-#{cert_filename}-certificate.ps1"
2  $import_script_name = "import-#{cert_filename}-certificate.ps1"
3
4  # Override scripts location and names defined by sslcertificate module.
5  # This will produce script files without the password in them.
6  File <| title == "${inspect_script_name}" |> {
7    path => "${scripts_dir}\\${inspect_script_name}.withoutteigi",
8  }
9
10 File <| title == "${import_script_name}" |> {
11   path => "${scripts_dir}\\${import_script_name}.withoutteigi",
12 }
  
```

Listing 15: Sample of *cernsslcertificate* module showing the use of resource collectors.

When using it in a manifest, the lookup and replacement of the key by the password is only occurs at the agent when it runs applies the catalog. Before we started to configure WTS using Puppet, it was being used mainly to configure Linux machines, so there was no implementation of *teigi*'s module for Windows. In Puppet each resource can have different implementations, written in Ruby - called providers - for different operating systems in order to offer a unique declaration syntax, thus reducing the developer's (or SysAdmin's) effort to configure different platforms. However, some properties may have

different behaviours, depending on the OS.

In *teigi*'s case, there was no provider for Windows systems for a particular resource - the sub-file resource (*teigi::secret::subfile*) - that could solve the password problem found in the *cernsslcertificate* module. To fix it, we had to develop a new provider for Windows; the code was, naturally, based on the existing Linux provider.

Since the Linux version of the resource also had attributes to manage file permissions using a Unix-like notation (owner/user,group,other), it required some modifications to work with Windows. Furthermore, the corresponding resource manifest - *teigi\_subfile.pp* had to be changed to contemplate the new provider, since the permissions had to be different. The full provider code is listed in annex XVI and the improved version of the *teigi\_subfile* manifest is detailed in annex XVII.

#### 4.4.5 Building Puppet types from DSC modules

Sometimes, e.g., when a new DSC module must be “imported” into Puppet, or some DSC modules have been updated or a new version of the *puppetlabs-dsc* module has been released, one must do a build to (re-)generate new Puppet types.

The *puppetlabs-dsc* module was developed by Puppet (the company was previously called Puppet Labs [127]) to generate Puppet types (resources) based on PowerShell DSC modules so that these modules can be used as Puppet resources in manifests (note that, in Puppet, the words *types* and *resources* can be used interchangeably).

In DSC, a module usually comes with various resources; for developers, these resources are similar to other Puppet resources, the only syntactic difference is that their name and their attributes start with “dsc\_”.

The *puppetlabs-dsc* module is delivered with some of the most used DSC modules (from Microsoft) already included, but it offers an instruction guide on how to build new Puppet types from DSC modules (either from Microsoft or the community) which, in the context of *puppetlabs-dsc* are called custom types.

The build is to be performed on a Linux machine, and the build environment requires some software packages - although, not all the packages listed in the guide are really necessary; we had to figure them out by iterating the building process multiple times until we found those really needed. The process benefits from having a purposely built Linux host, with a clean environment, therefore we have created an OpenStack VM and configured it (using Puppet) for that task.

However, we had some difficulties when configuring the VM its environment due to two main reasons: 1) Ubuntu was the best distribution to support the building process, mainly because of its package manager *aptitude* which had all the required software packages; 2) the Ubuntu image was not available in CERN's OpenStack private cloud. Conversely, the other Linux distribution images available on CERN's cloud (CentOS, Scientific Linux and Fedora) required a more complex installation of the software packages

necessary for the build environment, and they come with pre-loaded software that is not needed for the builds.

For those reasons and because the building process is not done frequently, we decided to try a different approach: create a Docker container [79]. This Docker container (and the file that creates it - Docker file) basically defines which OS image the container should be based on, and which packages should be deployed, thus creating a clean software environment for the building process. This Docker file is described in listing 16 and the project repository is available online [128] as a contribution to the open source community.

---

```
1 FROM ubuntu:16.04
2
3 WORKDIR /root
4
5 # Update packages
6 RUN apt-get clean && apt-get update
7
8 # Install locale to build DSC modules
9 RUN apt-get install locales
10 RUN locale-gen en_US.UTF-8
11 # Set locale environment variables
12 ENV LANG=en_US.UTF-8
13 ENV LANGUAGE=en_US.UTF-8
14 ENV LC_ALL=en_US.UTF-8
15
16 # Essential packages
17 RUN apt-get install git ruby ruby-dev build-essential libicu-dev libz-dev -y
18 RUN gem install bundler
19 RUN gem install semantic_puppet
20
21 # Non-essential but useful
22 RUN apt-get install vim tree curl -y
23
24 # Clone puppetlabs-dsc repository
25 RUN git clone https://github.com/puppetlabs/puppetlabs-dsc.git
```

---

Listing 16: Docker file to create a container to build Puppet types from DSC modules [128].

Two of the community DSC modules that were built with this module were: *xSystemVirtualMemory* and *PolicyFileEditor*. The first is intended to manage page file while the second is to manage local Group Policy rules. Listing 17 shows how these DSC resources are used in a puppet manifest after their puppet types have been built. Note that *dsc\_cadministrativetemplatesetting* is a resource part of the *PolicyFileEditor* module and *dsc\_file* is a resource pre-included in *puppetlabs-dsc* module.

A final note about the *puppetlabs-dsc* module: its design is not modular enough, since a new version requires rebuilding any previously built DSC modules. Nevertheless, it works well and its developers have plans to separate it in two (the module's core code base, and another module dedicated to custom types) [129].



```
1 class hg_windows_dev::dsc_tests {
2
3   # test default puppet resource
4   file{'c:/puppet_file_resource.txt':
5     ensure => present,
6     content => 'This is a test! Using Puppet File resource.'
7   }
8
9   # test simple resource
10  dsc_file { 'create-file-DSC':
11    dsc_ensure      => 'Present',
12    dsc_destinationpath => 'c:/puppet_dsc_file.txt',
13    dsc_contents    => 'this is a test! Using DSC through Puppet'
14  }
15
16  # test a custom resource - Set a Group Policy rule
17  # Remote Desktop Services\Remote Desktop Session Host\Licensing\Use the specified''
18  dsc_administrativetemplatesetting { 'test-dsc_cAdministrativeTemplateSetting License
19    ↪ Server':
20    dsc_keyvaluename => 'SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
21    ↪ Services\\LicenseServers',
22    dsc_policytype   => 'Machine',
23    dsc_data         => 'server.test.com',
24    dsc_ensure       => 'Present',
25    dsc_type         => 'String',
26  }
27
28  # test a custom resource - set page file
29  dsc_xsystemvirtualmemory { 'set-pagefile-with-dsc':
30    dsc_configureoption => 'CustomSize',
31    dsc_initialsize    => 2048,
32    dsc_maximumsize    => 4096,
33    dsc_driveletter    => 'C:'
34  }
35 }
```

---

Listing 17: Sample manifest using DSC resources

#### 4.4.6 WTS Puppet manifests

The manifest that configures the Remote Desktop servers that compose the WTS infrastructure is a product of the incremental configuration settings needed and requested over time for these machines. Hence, the manifest (see annex XVIII) shows the use of different resources and more complex configurations composed by multiple resources.

It's also possible to observe that there is a number of *exec* resources directly invoking PowerShell code. The reason is that for some of the configurations there was no Puppet resource to do it and they could be implemented by a few PowerShell commands inside the *exec* resource; after all, this is what the *exec* resource is for.

One example of configurations that require multiple resources is the synchronisation of the folder structure with special permissions (see lines 126 to 147). The goal of this configuration is to have a folder and its contents (stored in a file server) properly sync'd with correct permissions on the target machines.

At first we were using just the *file* resource and the *acl* resource - a Windows-specific module to set file permissions - since the *file* resource can be used to synchronise folders and *acl* to set the permissions on a folder and its sub-folders. But, in practice, this two-step strategy (involving two resources) was not working as expected: the folder structure (including sub-files) was being copied but the permissions were not being applied to the sub-folders, only to the top-level folder. After some experiments, we discovered that the problem was that the *acl* resource was not recursively setting the permissions as it was supposed and configured to do.

To solve this issue, we used a three-step approach: create a folder with the *file* resource; apply the permissions to the folder with the *acl* resource; and execute a synchronisation script with the *exec* resource. The reason why this approach works is because the copied files and folders inherit the top-level folder permissions. The synchronisation script was developed in PowerShell to verify if some file or folder has been changed w.r.t. the one stored in the file server, and in that case deletes everything and copies them again. The detailed script is described in annex XX.

Another manifest that we have developed, this one to configure a Remote Desktop License server, which shows how a DSC resource can be used to set-up a Group Policy setting, is described in annex XIX.

Furthermore, the modules described in the previous sections are present in this manifest since they help manage the desired configurations for the WTS infrastructure.

## AUTOMATION AND ORCHESTRATION

This chapter starts by defining automation and orchestration, as well as discussing some related concepts. Then, it presents two Microsoft tools, System Center Orchestrator and Service Management Automation (SMA), both designed to leverage automation and orchestration for Windows systems, and discusses their application to the WTS infrastructure. It concludes with a description of the work we have done with SMA in order to automate some important tasks in the CERN's WTS infrastructure.

### 5.1 Automation and Orchestration

Automation and orchestration are both essential in large-scale computing/cloud scenarios, as they help SysAdmins to manage infrastructures in a more flexible and efficient way by automating tasks in a programmatic way. The outcomes are: an infrastructure that can dynamically scale; reduced operational costs, that result from an increase in SysAdmins' efficiency, as they spend less time in manual interventions, and avoid errors.

Both concepts, Automation and Orchestration, share similar characteristics and benefits, but they are nevertheless different: automation addresses the individual tasks that must be accomplished (e.g. as formatting a disk, restarting a service or rebooting a server), while orchestration describes the arrangement and coordination of automated tasks, resulting in a consolidated process or workflow.

A (orchestration) workflow comes into play when such tasks must be done in a particular order or when the interaction with multiple entities is required (e.g. calling other services' APIs, registering servers in different services or storing data in multiple databases). Orchestration is also used to perform concurrent (may be in parallel) operations on multiple targets.

### 5.1.1 Runbooks and Workflows

Runbooks [130, 131] describe a combination of procedures and operations that need to be carried out by SysAdmins to accomplish a specific goal (e.g. install a software package, configure a system in a specific way, perform maintenance procedures).

The runbook concept comes from a past where instructions and procedures required to properly deploy, configure or operate a system had to be written in a physical book form. Procedures can be, e.g., start, stop, supervise, and debug a system; but they can also describe contingencies and the way special requests must be handled.

Runbooks are a way of describing how to manage a system, allowing SysAdmins (other than those who created the runbook) with an appropriated level of expertise to effectively operate it. Through runbook automation, these processes can be carried out using software tools in a programmatic manner. In Microsoft's terminology, runbooks are files that contain a sequence of operations which can be described in the form of workflows (graphical and/or script-based) or scripts (e.g. PowerShell, Python and Bash) [132]. Microsoft's use of the term runbook started with the introduction of the Orchestrator product, which allows SysAdmins to describe, using graphical workflows, the operations to be carried out (see fig. 5.1).

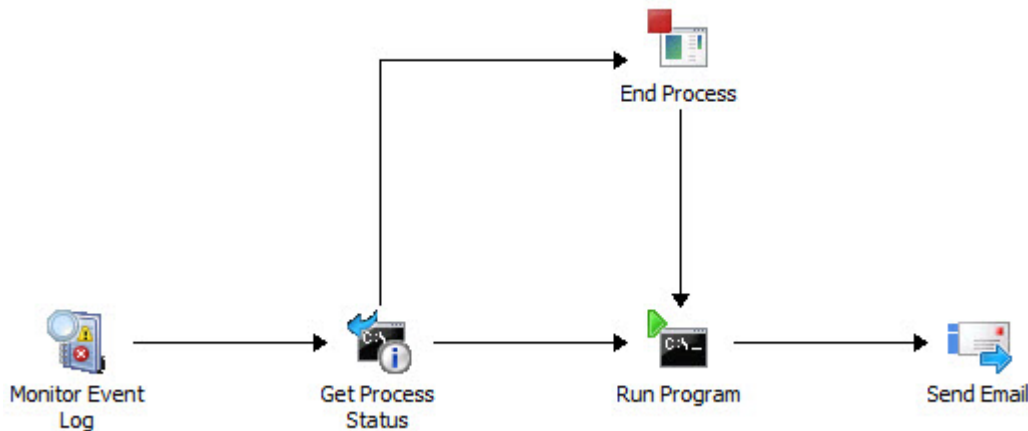


Figure 5.1: Orchestrator runbook example [133].

## 5.2 Evaluated Automation & Orchestration tools

For the WTS infrastructure, the main goals are automation and orchestration of frequent operations carried out by SysAdmins, up to the point of a full provisioning of VMs - which includes the deployment of the Puppet Agent (see fig. 5.2).

To orchestrate the WTS, two Microsoft products were evaluated: System Center Orchestrator and Service Management Automation. The two main reasons for this choice

were: a) they were designed for Windows systems and, b) they are available through the existing licensing contract with Microsoft.

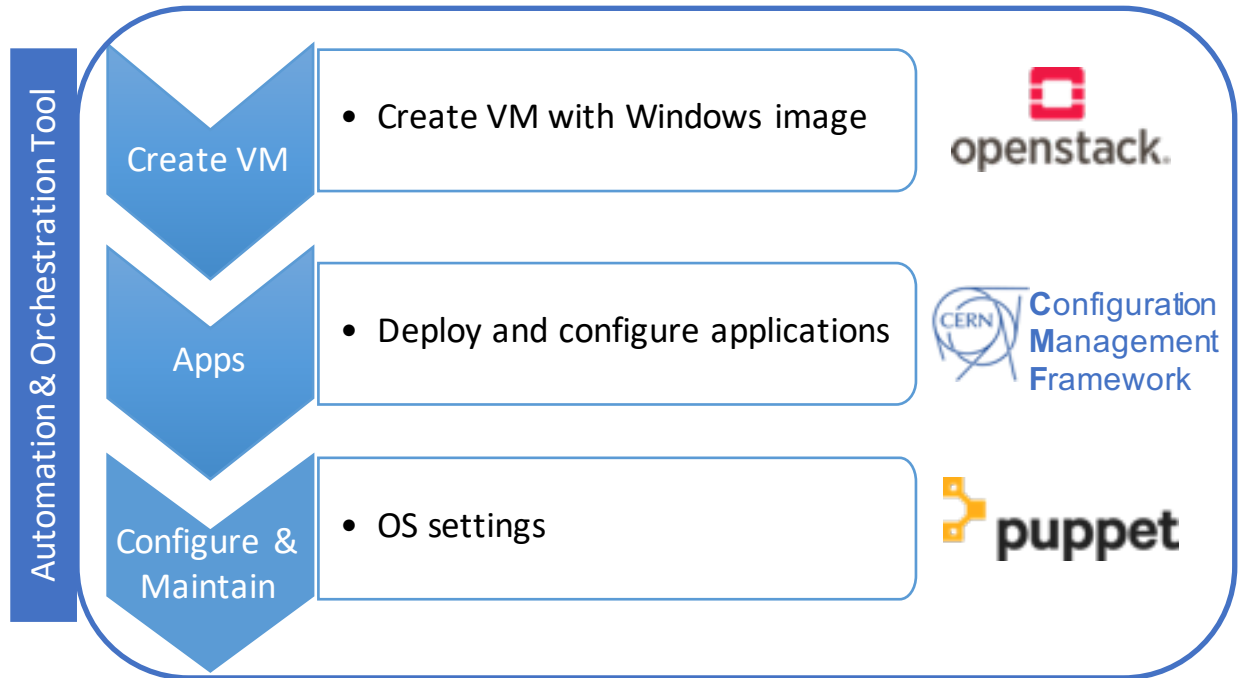


Figure 5.2: Provisioning steps of a VM in the WTS infrastructure.

### 5.2.1 System Center Orchestrator

Microsoft System Center Orchestrator (SCORCH) [13] or, in short, Orchestrator, is a software product designed to automate repetitive or error-prone processes across the data centre (DC), interacting, if needed, with multiple entities (both hardware, software and services), in a way that the reliability of DC operations is improved. Orchestrator is a standalone product but may be integrated with other services/tools to extend its functionalities.

Orchestrator allows SysAdmins to automate, with Orchestrator runbooks, repetitive or error-prone processes. Orchestrator runbooks are conceptually similar to scripts, in the sense that they perform some set of operations in a sequential and repeatable manner; the difference is that runbooks can be created by users that do not have a deep background in scripting or programming; however, in more advanced scenarios, runbooks can also include script components [134]. In Orchestrator, runbooks are designed through a drag-and-drop GUI and then translated into .NET, PowerShell, or SSH commands to ultimately automate tasks.

With Orchestrator it is possible to manage various System Center (the software suite Orchestrator is a part of) components, as well as Active Directory, and automate tasks on other OSs through the use of software modules (a.k.a. Integration Packs) [135].

Orchestrator's architecture is composed of multiple components (adapted from [136]):

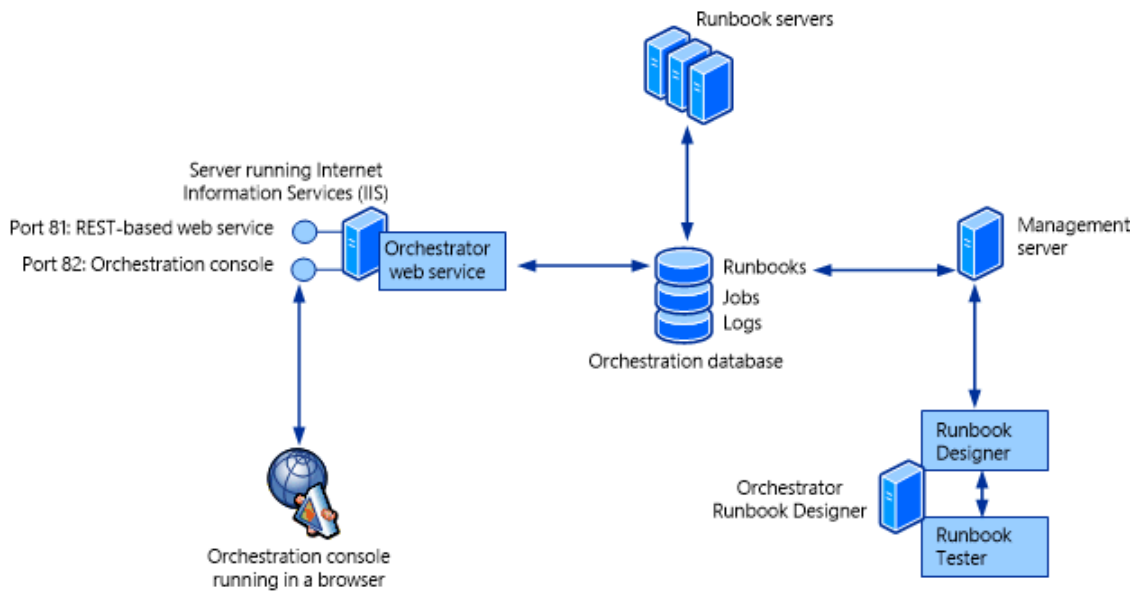


Figure 5.3: Orchestrator architecture [133].

- **Runbook Designer** is the GUI tool for creating and editing runbooks. A sub-component of the Runbook Designer is the Runbook Tester, which is used to validate the execution of runbooks.
- **Orchestration Database** is a Microsoft SQL Server database which stores runbooks, their status and security delegation configuration. The database also stores the log files and the configuration used in the Orchestrator deployment.
- **Management Server** is the core communication component of this architecture and is responsible for coordinating the communication between the Runbook Designer and the Orchestration database. There is only one Management Server per Orchestrator deployment.
- **Runbook Server** is responsible for executing instances of runbooks. When a runbook is invoked, a copy (instance) of the runbook is sent to its assigned Runbook Server and then executed. The first installed Runbook Server is assigned the Primary role.
- **Orchestrator Web service** is the interface that enables applications to connect to the Orchestrator. Typical tasks performed through this Web Service are runbook status views, and start and stop actions.
- **Orchestrator Browser Console** is a Silverlight [137] supported web browser which uses the Orchestrator Web Service to communicate with Orchestrator.

Summarising, Orchestrator helps SysAdmins to [138]:

- Automate processes and enforce best practices for incident, change, and service-life-cycle management;
- Reduce unanticipated errors and service delivery time by automating tasks across responsibility groups within the IT organisation;
- Integrate non-Microsoft tools to enable interoperability across the data centre;
- Orchestrate tasks across systems for consistent, documented, and compliant activity.

### 5.2.2 Service Management Automation

Service Management Automation (SMA) is a software package delivered in the Windows Azure Pack [139] - a software suite that brings Microsoft's cloud technologies to the data centre, to Windows Servers. Like Orchestrator, it enables SysAdmins to create, run, and manage runbooks, helping them to automate and orchestrate IT business processes. It is based on Azure Automation - the cloud management automation solution for Microsoft's Azure IaaS - but targeted to on-premises data centres.

Both SMA and Azure Automation use runbooks implemented as Windows PowerShell Workflows whereas Orchestrator uses graphical runbooks. SMA follows the IaC approach as it enables the management and automation of computing infrastructures using code.

A Windows PowerShell Workflow is similar to a PowerShell script but has some significant differences: a workflow is a sequence of programmed, connected steps, that perform long-running tasks or require the coordination of multiple activities (steps) across multiple managed nodes. The benefits of a workflow over a normal script include the ability to simultaneously perform an action against multiple nodes and the ability to automatically recover from failures. A more detailed explanation about PowerShell workflows is described in section 5.3.

Windows PowerShell Workflows are well suited for executing tasks that need:

- To run for an extended period of time;
- To run in parallel for increased efficiency;
- To survive reboots and disconnected sessions;
- To be suspended and resumed without loss of data;
- To be throttled or connection-pooled in large-scale or high-availability environments.

Another feature of SMA is the Global Assets store, a central store capable of saving and retrieving assets such as variables, credentials, certificates and connections [140]. This store is useful to share static data among runbooks and enhances security by avoiding sensitive information in the code (e.g. plain-text credentials).

The SMA architecture is composed of multiple components (see fig. 5.4):

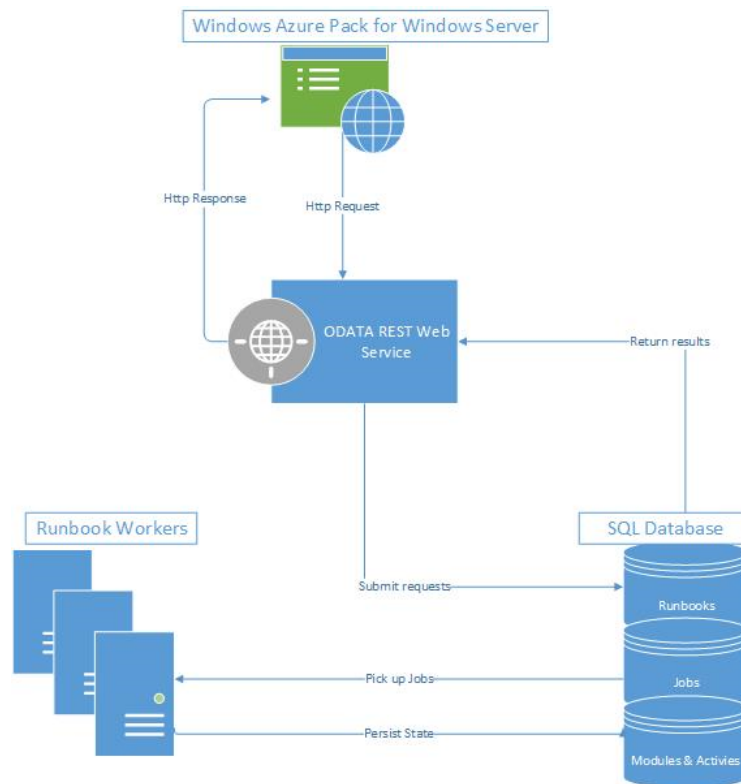


Figure 5.4: SMA architecture [14].

- **Automation web service** which connects to Windows Azure Pack, authenticates users and distributes runbook jobs to runbook workers.
- **SQL Server databases** that allow storage and retrieval of runbooks, runbook assets, activities, integration modules, and runbook job information.
- **Runbook workers** to run the runbooks, and can be used for load balancing.
- **Management Portal** the web portal where runbooks are created, debugged, started and stopped.

### Dissecting runbook execution

When the SMA web service makes request - either through the Service Management Portal or the *Start-SmaRunbook* PowerShell cmdlet (CLI command) - to start a runbook execution, the web service creates a job - an runbook execution with the provided arguments - and writes it to the SMA database for subsequent retrieval by one of the runbook worker servers.

A runbook worker server runs a job, picked from the database, and remotely accesses any computer or other resources that it needs to work with, as specified by the runbook code (e.g., execute commands on remote computers). When a job is suspended or interrupted, it may be resumed on a different runbook worker, as the state of the job is recorded on the database server.



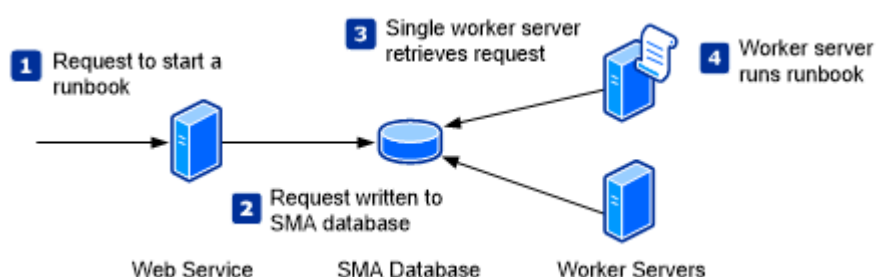


Figure 5.5: SMA runbook execution steps [14].

SMA uses static queue partitions to load balance work among runbook workers and, because of this, it cannot adjust its job scheduling when new workers are added, removed or go offline. If a worker in the current deployment goes offline, it is still allocated jobs but it cannot process them. Similarly, if a new worker, not configured in the current runbook worker deployment - the configuration that defines all the workers that constitute the SMA infrastructure - is started, it will not be assigned any jobs.

### Choosing the Orchestration Tool for CERN

For the WTS infrastructure, the team has chosen to adopt Service Management Automation (SMA) as the tool to help manage, automate and orchestrate processes and minimise the operational costs - mainly time and human labour - of the infrastructure. System Center Orchestrator was considered but it had several drawbacks: a) it doesn't offer the same set of functionalities nor the IaC approach (and SMA does); b) Microsoft has made public its intention to stop supporting and developing Orchestrator, as the new designs are IaC- and PowerShell-based and target both the data centre and Microsoft's IaaS Azure cloud.

Another, also important reason for choosing SMA is that the WTS team members are very familiar with the PowerShell syntax as it has been used to develop scripts that helped automate some operational tasks.

## 5.3 Windows PowerShell Workflows Concepts

In order to better understand the work done with SMA and runbooks, it is important to introduce the concepts behind Windows PowerShell Workflows, which were released with the Azure Automation tools, for Azure, and with SMA, for on-premises data centres.

A Windows PowerShell Workflow (or just workflow, for short) is a script containing a sequence of programmed steps that perform long-running tasks and/or require their execution to be coordinated across multiple nodes. In terms of syntax, a workflow starts with the *workflow* keyword, along with its name (also referred as the name of the runbook), followed by the body (that looks like a PowerShell script) enclosed in braces. A workflow's body is composed of one or more tasks, called activities, that are executed sequentially;

the code in listing 18 depicts the basic structure of a Windows PowerShell Workflow with parameters.

---

```
1 Workflow Test-Runbook
2 {
3     Param
4     (
5         [Parameter(Mandatory=<$True | $False>)]
6         [Type]$<ParameterName>,
7
8         [Parameter(Mandatory=<$True | $False>)]
9         [Type]$<ParameterName>
10    )
11    <Activities>
12 }
```

---

Listing 18: Windows PowerShell Workflow basic structure (adapted from [141]).

A workflow (a.k.a runbook) file has a .ps1 suffix and is processed by Windows Workflow Foundation (WWF) “engine” [142, 143]. This shows the close ties between runbooks and PowerShell scripts: the syntax and structure of is almost the same. And, to tighten things up even more, there is a large set of PowerShell cmdlets that are available as activities in WWF (however, some of these activities may be slightly different from their corresponding cmdlets, e.g., less available options) [144]. The *InlineScript* construct, however, allows any PowerShell cmdlet to be used; but their execution engine is now a PowerShell-session process triggered by WWF’s engine (see subsection 5.3.1 for details).

In a workflow, each activity runs in its own session (a.k.a. runspace) in the workflow process and the session management is done by Windows PowerShell Workflow itself; activities cannot therefore share data, such as variables created in the session; however, sessions do inherit all variables in the top-level workflow scope.

It is not possible to invoke methods in a workflow due to its design: the activities in workflows are converted to XAML and they return serialised (XML-formatted) representations of objects. These objects have properties and property values, but the methods are not available; nevertheless, workflows can define and then call functions which are handled like other cmdlets. Functions may be declared the workflow scope or in PowerShell modules (a.k.a. Integration Modules); functions defined in a workflow are not visible by other SMA workflows. Code listing 19 depicts an example of a workflow calling a function.

Integration Modules are packages containing PowerShell modules that provide sets of cmdlets that can be imported to SMA in order to be invoked from runbooks.

There are plenty of modules developed by Microsoft and the developer community, available in a repository called PowerShell Gallery [145]. A developer can author its own module and import it to SMA, to automate some task or encapsulate a set of functions that are frequently used by multiple runbooks. Once a module is imported into the Automation server, it will be available to all Runbook Worker servers, which can then

```
1 Workflow Test-Runbook
2 {
3     Test-Function -Name "test" # Calling function Test-Function
4
5     function Test-Function ([string] $Name) {
6         return $Name
7     }
8 }
```

---

Listing 19: Windows PowerShell Workflow calling a function.

execute it. Runbooks don't need to explicitly include modules because, when executing a command/activity, modules are automatically loaded by the Windows PowerShell Workflow,

### 5.3.1 InlineScript

The InlineScript activity runs a block of PowerShell commands in a separate, non-Windows Workflow Foundation session, and returns its output to the workflow. While most commands in a workflow are sent to WWF for processing, commands in an InlineScript block are processed by PowerShell.

The commands in an InlineScript script block run, by default, in a single session and can share data, such as the values of variables.

The InlineScript activity supports a set of parameters, named Workflow Common Parameters; as an example, the *PSComputerName* and *PSCredential* parameters allow to specify that the script block must run on another host and specify the credentials to execute the code. The syntax of this activity is depicted in the code listing 20.

```
1 InlineScript
2 {
3     <Script Block>
4 } <Common Parameters>
```

---

Listing 20: InlineScript syntax [141].

The most common use for the InlineScript in a workflow is to run a block of code on another computer (see fig. 5.6). This is ideal for two scenarios where activities can't be used: when cmdlets in the runbook belong to the excluded set of cmdlets that can run in workflows, or when there is a sequence of commands that must be performed locally on the target host.

Code listing 21 shows an example of a workflow that connects to a remote host and runs a set of commands that format any disk drive that is not initialised (raw). In this example, a user credential is obtained from the SMA Global Asset store and is used as a parameter to the InlineScript activity so that the code block can run with the required

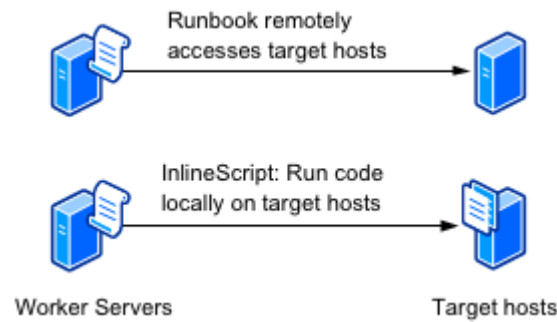


Figure 5.6: Execution of PowerShell commands on remote computers using InlineScript [141].

level of privileges. Inside the InlineScript block, there is a pipeline of PowerShell commands that fetches the available disks in the remote computer, selects the ones that are not initialised (raw), initialises and formats them to a NTFS filesystem.

---

```
1 Workflow Test-Runbook
2 {
3     $Cred = Get-AutomationPSCredential -Name 'adminuser'
4
5     InlineScript
6     {
7         Get-Disk | Where-Object partitionstyle -eq 'raw' |
8         Initialize-Disk -PartitionStyle MBR -PassThru |
9         New-Partition -AssignDriveLetter -UseMaximumSize |
10        Format-Volume -FileSystem NTFS -NewFileSystemLabel 'DATA' -Confirm:$false
11    } -PSComputerName 'server01.mydomain.com' -PSCredential $Cred
12 }
```

---

Listing 21: Formatting a new volume in a remote computer using an InlineScript [141].

The common parameter *PSComputerName* can have more than one computer name, allowing the same code to be executed in multiple hosts in parallel. The computer names are specified using a string separated by commas (e.g. “server01,server02,server03”).

The InlineScript activity can access variables from the workflow scope using the **\$Using:** prefix for each variable. The output returned by the InlineScript block can be assigned to a variable. Code listing 22 illustrates how to pass and return values from an InlineScript activity.

Although an InlineScript activity has many advantages it also has limitations: a) as it is processed by the PowerShell engine, it is not possible to run other workflow activities inside the InlineScript block; b) in case of a failure, it is not possible to resume the InlineScript activity, only to restart it from the last checkpoint or from the beginning of the runbook; and c) it hinders the scalability of the runbook as the execution of other activities is held until the InlineScript block is finished.

```
1 Workflow Test-Runbook
2 {
3     $data = "example"
4     $output = InlineScript
5     {
6         Write-Output "This is an $Using:data"
7     }
8 }
```

---

Listing 22: Passing and returning values from an InlineScript activity.

### 5.3.2 Checkpoints

Windows PowerShell Workflows have a persistence mechanism that helps them survive failures by using checkpoints. A checkpoint is a snapshot of the current state of the workflow that includes the current value for variables and any output generated to that moment.

Checkpoint data is stored in the SMA database and preserved until another one is taken, in which case the first checkpoint is overwritten, or until the runbook completes (workflow finishes its execution). Checkpoints are taken by invoking the *Checkpoint-Workflow* activity which immediately takes a snapshot and stores it in the database.

If a problem occurs (e.g. power outage) and the processing of the workflow is interrupted, it can be resumed again near the point of interruption. A checkpoint also ensures that an action will not occur more than once and have a negative effect - idempotency. Also, if a runbook's execution is suspended due to an error, when the job is resumed, it will restart from the last checkpoint.

Listing 23 is a sample code where an error occurs after Activity2 causing the runbook to suspend. When the job is resumed, it starts by running Activity2 since this was just after the last checkpoint set.

```
1 Workflow Test-Runbook
2 {
3     <Activity1>
4     Checkpoint-Workflow
5     <Activity2>
6     <Error>
7     <Activity3>
8 }
```

---

Listing 23: Checkpoint-Workflow example.

Checkpoints should be taken after activities that may be prone to error and should not be repeated in case of a failure. For example, if a runbook creates a VM checkpoints could be taken both before and after the commands to create the VM, so that the if the creation has failed, a new attempt can be made by resuming the runbook job from the

first checkpoint; and, if the creation was successful but the runbook job failed afterwards, the job restart would not try to create the VM again.

There is some overhead associated to checkpoints, as they require that the data that represents the current state to be serialised and stored in the database. Hence, if there are operations (chunks of code) that take less time to repeat, are not critical and are idempotent, the use of checkpoints should be carefully planned as they can negatively impact the overall runbook execution time.

### 5.3.3 Parallel Execution

Windows PowerShell Workflows engine has the ability to execute a set of commands in parallel; this feature is particularly useful in runbooks as it allows to concurrently execute multiple actions that take a significant time to complete, increasing the overall efficiency. For example, a runbook could provision a set of VMs in parallel, rather than sequentially, and continue to the next action only after all VMs were provisioned.

In workflows there are two ways to execute a set of commands in parallel: using the *Parallel* block or the *ForEach -Parallel* loop.

The *Parallel* block allows multiple commands inside the block to run concurrently. As code listing 24 shows, *Activity1* and *Activity2* will start at the same time whereas *Activity3* will start only after both *Activity1* and *Activity2* have completed.

---

```
1 Workflow Test-Runbook
2 {
3     Parallel
4     {
5         <Activity1>
6         <Activity2>
7     }
8     <Activity3>
9 }
```

---

Listing 24: An example using the *Parallel* block.

If a set of commands needs to be executed sequentially inside the *Parallel* block, then the *Sequence* block must be used. The *Sequence* script block runs in parallel with other commands, but the commands within the block run sequentially. An example that illustrates this behaviour is depicted in code listing 25 where *Activity1*, *Activity2*, and *Activity3* will start at the same time; *Activity4* will start only after *Activity3* has completed; and *Activity5* will start after all other activities have completed.

The *ForEach -Parallel* loop is used to process commands for each item in a collection concurrently. The items in the collection are processed in parallel while the commands inside the script block run sequentially. Code listing 26 shows an example using the *ForEach -Parallel* loop. In the example, *Activity1* will start at the same time for all items in the collection and, for each item, *Activity2* will start only after *Activity1* is complete.

```
1 Workflow Test-Runbook
2 {
3     Parallel
4     {
5         <Activity1>
6         <Activity2>
7         Sequence
8         {
9             <Activity3>
10            <Activity4>
11        }
12    }
13    <Activity5>
14 }
```

---

Listing 25: An example using the Parallel block with a sequence of commands.

Finally, Activity3 will start only after both Activity1 and Activity2 have completed for all items.

```
1 Workflow Test-Runbook
2 {
3     ForEach -Parallel ($<item> in $<collection>)
4     {
5         <Activity1>
6         <Activity2>
7     }
8     <Activity3>
9 }
```

---

Listing 26: An example using the ForEach -Parallel loop.

## 5.4 Service Management Automation usage at CERN

SMA was adopted and deployed by the WTS team to automate and orchestrate many operational tasks of the infrastructure. First, SMA was deployed as a proof-of-concept (PoC) in a single server running Windows Server 2012 R2 with all SMA components (automation web service, database and runbook worker). In this stage, it was only used in a development environment and a few runbooks (using Windows PowerShell Workflows) were developed to get familiar with the software and to explore its capabilities and limitations, if any.

Later on, two identical infrastructures were deployed to serve two different environments: development and production. The development environment is used to develop and test the runbooks against test servers - disposable VMs only used for testing, while the production environment is where stable/tested runbooks are executed against the production servers that form the WTS.

More recently, the development infrastructure was rebuilt using Windows Server 2016 and two VMs: one with all components, and the other as an additional runbook worker. The reasons for this approach were: a) to adopt the latest Windows Server version, since it provides more PowerShell cmdlets; and, b) to evaluate overall stability and performance of SMA running under the new OS and with an additional, separate, runbook worker.

In both infrastructures, the database is not stored in a VM “local”, disk but in a Ceph volume attached to the VM, to provide a simple form of fault tolerance (in case the VM fails, the volume can be attached to another VM to ensure that the SMA service can be quickly recovered) or if a server migration takes place (thus preserving the database contents).

Although Microsoft recommends to have 3 runbook workers and a separate database server [146], the infrastructures previously described was deployed with a minimalist mindset by using a few but enough computing resources (VMs on OpenStack and storage on Ceph) to run the existing runbooks.

With time, as more runbooks are developed to automate operations, if we find them more computing intensive, more resources will be added. Despite SMA’s adoption is still in an early stage, the plan is to scale up the infrastructure as SMA’s workload increases. The team’s plan is to have a production infrastructure with three runbook workers and two database servers for redundancy - a primary database server and a second fail-over server.

## 5.5 Our Work with the SMA

During our stay with the CERN’s WTS team, a few runbooks were developed to increase the level of automation and orchestration of the infrastructure; in this subsection we describe two different contributions to this project: an Integration Module and a runbook to create VMs for the WTS.

### 5.5.1 CERNOperations Integration Module

As explained in section 5.3, an Integration Module is a PowerShell module that defines functions that can be imported into SMA and used in runbooks.

The CERNOperations module contains functions that perform many different operations and communicate with different systems. The module performs operations such as testing if a server is reachable (connect to host), send notification e-mails, delete a VM or a volume on the OpenStack cloud infrastructure, check status and trigger actions on a VM’s CMF agent, etc.. The majority of these functions do interact with target hosts, either to retrieve some data, or perform some operation; as such, they are a good showcase of the orchestration capabilities of SMA.

As the module’s code is quite long, it’s impracticable to discuss everything here, so please refer to annex XXI. Instead, we focus on some of the functions that are relevant to



understand the *create-vm-with-volume* runbook which, as the name implies, creates a VM with an attached volume.

### WaitForVM

We start by describing the *WaitForVM* function which waits for a Windows VM hosted on OpenStack to finish its provisioning process and be remotely accessible. This function was designed to encapsulate the loop that waits for a VM while it is being provisioned, so that runbooks like *create-vm-with-volume* and others in the future can use it before starting to invoke commands on the VM.

The function code has two phases (see fig. 5.8, each one characterised by a busy-waiting loop: the first loop waits for the VM to reach OpenStack’s running state (a.k.a. “ACTIVE” state), after having been successfully created and powered up; the second loop waits for the Windows OS to be fully installed and able to service remote commands through WinRM. Both loops check the status of the VM’s OpenStack and WinRM states and, if they haven’t reached the desired level, sleep for a given time and retry.

In the first phase, the current state of the VM is retrieved using a CLI-based command that calls OpenStack’s API. This is accomplished through services made available by the CERN’s dedicated management cluster, called AIADM (which stands for Agile Infrastructure Administration), which has all tools (e.g., a set of Python CLIs developed at CERN called *ai-tools* [147]) that interact with other services’ APIs (like OpenStack, Foreman and Puppet) and administer managed servers in the infrastructure.

Hence, to make a request to OpenStack from the PowerShell function, the *Invoke-SSHCommand* cmdlet is used to invoke a command on an AIADM server through SSH. Once the administration server executes the command, the result is returned by the *Invoke-SSHCommand* (see fig. 5.7).

Note: the *Invoke-SSHCommand* cmdlet is part of a Integration Module called SSH [148] that had to be imported into SMA.

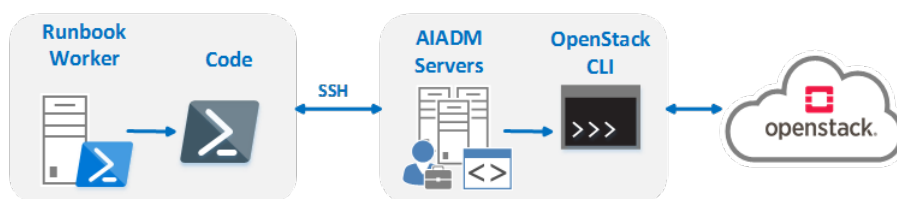


Figure 5.7: Making a request to OpenStack through AIADM servers.

The commands that our module invoke on AIADM perform both a *set* and a *get*: they set the OpenStack project where the VM is located, and get the VM’s state.

In the second phase, the *Test-WSMan* cmdlet is used to check if the VM is reachable through WinRM. By default, Windows Server (since version 2012) have this service enabled, thus it makes sense to wait for the operating system to be fully deployed and WinRM service ready, so that the VM can execute commands sent through WinRM. The

function returns a null object in case the machine is not reachable via WinRM, otherwise an object with the WinRM state is returned.

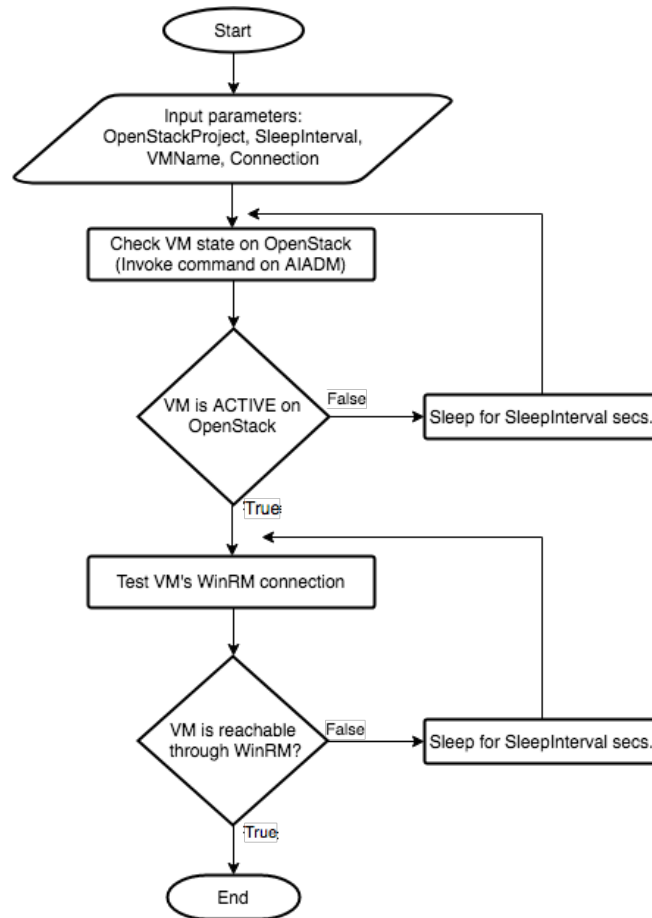


Figure 5.8: Flowchart of the *WaitForVM* function.

### ***WaitForCMF***

The *WaitForCMF* function waits for the CMF agent to finish installing all the software applications on the deployed Windows server VM. It is, therefore, similar to *WaitForVM*.

As previously referred in section 2.3.4, a) CMF provides a way of, based on computer groups (a.k.a. NSCs), distributing and installing applications on Windows systems, and b) Windows images available on CERN's OpenStack have the CMF agent pre-installed so that, after the OS is installed and the VM boots up, applications can be installed automatically.

As the CMF agent reports all the status changes to the CMF web service, we use the web service SOAP-based API, in particular the *CMFCompleted* method, to ask for the last reported status of the VM.

Therefore, similarly to *WaitForVM*, a busy-wait loop is used to wait until the CMF agent reports that all the pending installations were concluded (see fig. 5.9). Figure 5.10 illustrates the request made from *WaitForCMF* function's code to the CMF web service to get the VM's CMF agent status report.

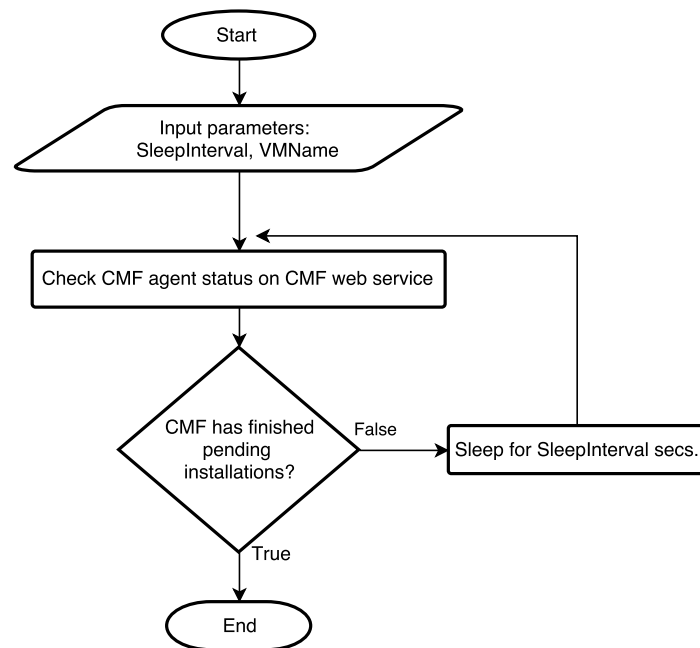
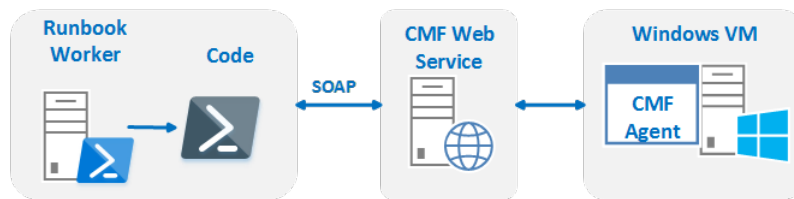
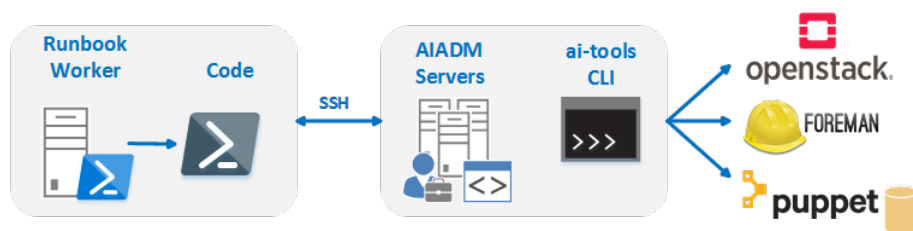
Figure 5.9: Flowchart of the *WaitForCMF* function.

Figure 5.10: Contact CMF Web Service to get the status of a VM's CMF Agent.

### DeleteVM

The *DeleteVM* function deletes a VM on OpenStack and all its information on Foreman and Puppet DB. It invokes the *ai-kill* command on AIADM to trigger the deletion process: kills a VM on CERN's OpenStack private cloud, then deletes all its data on Foreman and Puppet DB (see fig. 5.11).

Figure 5.11: Invoking *ai-kill* to delete Puppet-managed servers on OpenStack.

### 5.5.2 Runbook *create-vm-with-volume*

One of the most frequent tasks the WTS team has to do is the provisioning of new VMs either to prepare new clusters or to replace existing ones. For this reason a major goal was to develop a runbook that automates the creation of Windows server VMs for the WTS infrastructure (see fig. 5.12); that was accomplished with the *create-vm-with.volume* runbook, which creates a single Windows VM on CERN's OpenStack, attaches a secondary volume, and ensures that Puppet is running.

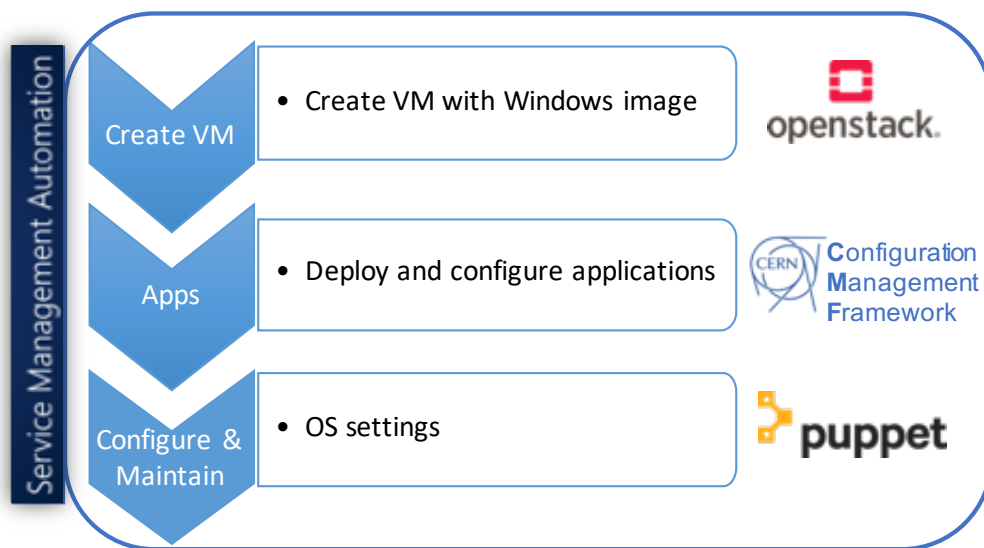


Figure 5.12: Orchestrating the provisioning process of a VM in the WTS infrastructure.

The *create-vm-with-volume* runbook opens an SSH session to an AIADM server and invokes the *ai-bs-vm* command from the *ai-tools* toolset, to create a Puppet-managed VM on OpenStack. The *ai-bs-vm-command* creates a VM on OpenStack and registers it on Foreman within a given hostgroup, thus defining where the VM's Puppet agent has to fetch configurations.

Furthermore, this command takes multiple arguments, but we highlight just the ones we use: the Foreman hostgroup; the responsible user/group for the VM; a string of OpenStack information about the VM - project where the VM has to be created, availability zone, flavor (hardware), OS image to install, and type and size of the volume to be created and attached; and some metadata that specifies the CMF NSCs (groups) the VM must belong in order to have the desired software applications.

The runbook's code can be found in annex XXII, while the flowchart of figure 5.13 depicts all the operations performed by the runbook:

1. Create a VM using *ai-bs-vm* tool with the input arguments;
2. Check if any errors occurred when creating the VM; if so, delete the VM and its volume using the delete functions of CERNOperations' module;
3. Perform a Checkpoint-Workflow to avoid re-creating the VM in case SMA fails;

4. Wait for VM to be up and able to service commands, using the *WaitForVM* function;
5. Wait for the CMF agent to finish the installation of all the applications, using the *WaitForCMF* function;
6. Perform a Checkpoint-Workflow before restarting the VM;
7. Restart the VM to reload all the operating system services and install the SSL certificates (generated by CERNs Certification Authority and distributed by Active Directory). The *Restart-Computer* cmdlet is used to restart the VM and wait for it to boot up and be reachable through WinRM;
8. Perform a Checkpoint-Workflow before formatting the attached volume;
9. Format the attached volume to NTFS and label it "DATA";
10. Check if Puppet could find the SSL certificates; if not, fix the Puppet installation using the *FixPuppetCert* function from CERNOperations' module;
11. Restart Puppet Agent service to reload Puppet and force it to fetch configurations;
12. Notify all the WTS SysAdmins through e-mail that the VM has been created, is running, and Puppet will apply the desired configurations.

Before we had automated this task, it could take up to three hours just to install and configure applications on a single VM, and that was without performing any OS configuration; to that amount of time one has to add about thirty minutes, the time required to create the VM itself on OpenStack.

The team usually creates VMs in small batches (two or three) and then joins them to the corresponding clusters. SysAdmins, as humans in general, cannot multitask different jobs - here, creating VMs, installing and configuring applications - that require a lot of attention to details, or else mistakes do happen - and those are usually costly to recover from.

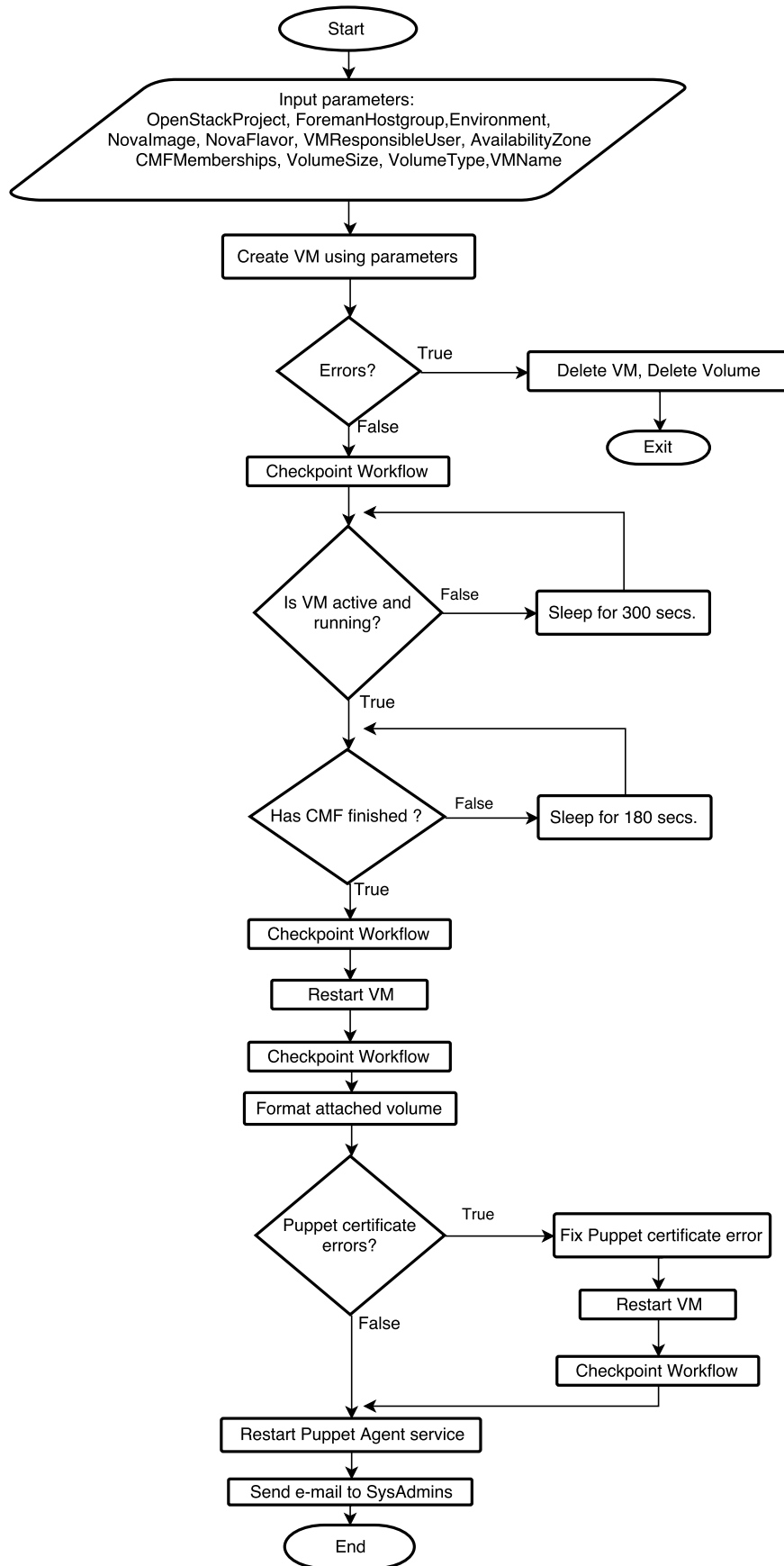
In this scenario, the complete provisioning, configuration and validation of a batch of servers was taking between one to two days (eight hour work days) - the validation procedure was a manual one, and required additional efforts to check if the required configurations and applications were in place. Thanks to the use of Puppet and CMF, this validation is now mostly done when new configurations are applied - either in testing phase or the first time they are used in production.

Thanks to the use of this runbook we were able to reduce the time to fully provision and configure - with Puppet and CMF - a VM to two hours and a half, including OS configurations. And if, after VM's creation, it happens that some operations cannot be automated, the time spent by SysAdmins on those tasks is minimal, when compared to the savings brought in by automation.

Overall, the runbook helped reduce both SysAdmin's manual interventions, error-prone operations and the total provisioning and configuration time of the WTS' VMs.

Although the *create-vm-with-volume* runbook only creates one VM at the time, it can be executed multiple times to create as many VMs as desired without SysAdmin intervention. As future work, another runbook may be developed to create multiple VMs simultaneously by calling this one, e.g., in a *ForEach -Parallel* loop.

Runbooks such as *create-vm-with-volume* illustrate the automation and orchestration capabilities of SMA, as they perform multiple operations on different systems to accomplish a single goal.

Figure 5.13: Flowchart of the *create-vm-with-volume* runbook.





## CONCLUSIONS & FUTURE WORK

### 6.1 Conclusions

With a team as small as three people, an infrastructure with the size of the WTS (about 150 servers and growing) cannot be effectively managed and maintained without the use of software tools that enable the automation of many operational tasks.

CERN's IT Department tries to adopt free and open source tools as much as possible to reduce costs, although it is not always an easy decision the choice between commercial (sometimes referred as enterprise versions), mostly proprietary versions of the tools, and free, open-source versions; IT managers have to analyse not only the product features, but the costs associated, including support and documentation. Sometimes it pays off to use enterprise tools because they have better support (help from specialised professionals), compatibility, integration, documentation and are more easily deployed and require less man-power (development or other tasks).

Since the IT Department started its Agile Infrastructure project, free and open source tools have been chosen, such as OpenStack (to operate CERN data centres and provide a private IaaS) and Puppet (to automate and manage configurations of VMs).

Although Puppet was the initial choice for the SCM tool, it was chosen at a time where there was essentially only another SCM tool, Chef, that provided the required features to manage CERN's cloud infrastructure. Both tools, Puppet and Chef were initially released to manage Linux/UNIX configurations; but, recently, they have been extended to support Windows systems as well.

Things have now changed and, more recently, other tools have appeared in this area of SCM and some of them are based on a different approach, while also supporting configuration management of Windows systems.

Therefore, we explored two other configuration management tools - Ansible and PowerShell DSC - to see if Puppet was still the best tool to automate and manage configurations on the Windows servers that compose the WTS infrastructure. As a result of our research and experimentation, we found that:

- Ansible is the easiest to set up and use, thanks to its design and the simplicity of the YAML language, but doesn't ensure compliance "out-of-the-box" - periodic configuration runs to avoid drifts from the desired configuration;
- PowerShell DSC, Microsoft's solution, offers plenty of resources designed to manage Windows systems. Based on PowerShell - a scripting language commonly used by Windows SysAdmins - it does not, however, provide the same level of management of hostgroups and configurations as the other SCM tools;
- Puppet, on the other hand, uses a language that is more difficult to learn, and its set up is more complex; but it has demonstrated to be a mature tool that is able to ensure compliance and manage configurations properly.

In the end, the WTS team chose to take advantage of the Puppet-based configuration management infrastructure already available at CERN, as it already has a dedicated management team which also provides support for other CERN users.

As consequence, Puppet manifests were developed to configure the Windows servers in the WTS infrastructure; these manifests had a good impact in the WTS team's effort to configure servers, as they greatly reduced manual configuration tasks and time spent to deploy servers with the desired configurations. In addition, multiple improvements were done in a few Puppet modules, and that has helped quite substantially the development of Puppet manifests.

Despite the disparity, in Puppet, between the number of modules designed for Linux and Windows, the use of the *puppetlabs-dsc* module, which enables the integration of DSC resources already available to manage Windows systems, greatly reduces that above-mentioned disparity. That integration is not easy but, nevertheless, it does provide a way of managing resources using the same syntax as in Puppet manifests, and no modules are required to be installed on the target machines as the agent downloads those required on-demand from the master.

Configuration management, "per se", is not enough to deliver the level of automation needed for the WTS infrastructure, as it just provides an automated way of configuring many servers. In order to perform maintenance tasks and provision customised Windows Server machines, another tool was needed for the automation and orchestration. So, Microsoft SMA was selected as it is based on PowerShell and enables the development of scripts that can perform actions in multiple machines at once.

By using SMA, we were able to create PowerShell workflows to provision Windows Server machines and execute some initial configuration tasks (done only once), such as installing the Puppet agent, or formatting a secondary storage volume. SMA provides

many features that make it a great tool for automation and orchestration tasks of Windows machines, such as: the capability of running very long jobs with checkpoints, so they can be resumed in case of an error that causes the job to be suspended; remote execution of commands and script-blocks (blocks of code) on multiple machines in parallel; detailed reporting and logging of the job's execution; a centralised credential storage (to avoid hard-coded passwords).

Before we started this project, it could take up to three hours just to configure and install applications on a single server, depending on the server's required configurations, plus the time needed to create a VM instance on OpenStack, which takes about thirty minutes. The normal scenario is to create batches of two to three servers simultaneously and then perform the required configurations and installations manually. As a SysAdmin cannot multitask as computers do, he/she just performs operations in some machines while waiting for others to finish. In this scenario, the complete provisioning, configuration and validation of a batch of servers was taking between one to two days (eight hour work days).

With the adoption of automation technologies, following a DevOps methodology, we were able to reduce the 3,5 hours to roughly two hours and a half, for a single machine, including the VM creation and the configuration (carried out by Puppet). Even if some tasks still have to be done manually or require SysAdmin interaction, the time spent on such tasks is minimal when compared to the time needed to perform repetitive tasks. The key advantage is that multiple identical servers can be provisioned simultaneously in an automated way.

Furthermore, in a large organisation where there are frequent staff changes (new staff members and departure of others) and small teams have to manage large infrastructures, like the WTS, it's crucial to have "recipes" to ensure that repetitive tasks are done properly and in an automated manner without relying on a specific person's knowledge.

## 6.2 Future Work

Initially, the vision to leverage automation and orchestration to the WTS was based on the integration of multiple software technologies: Puppet for configuration management and SMA as a central component to automate tasks that involved integration with other tools like Microsoft SCOM, for alarms and monitoring, and HAProxy to redirect the RDP connections when some servers are being decommissioned or maintained.

This initial vision for the WTS is reflected in a conference poster [149] where some use cases for SMA and PowerShell workflows were presented: actively check the connectivity of Remote Desktop servers and reboot them in case of no response; clean the profiles folder of a server when its disk has low space available (triggered by a SCOM alarm); execution of maintenance tasks such as installing updates and patches. The last two cases also illustrate the needed integration between SMA and HAProxy, as nodes need to be set to "drain" mode so that no RDP traffic is directed to them.

No PowerShell workflows have been yet developed to integrate SCOM and HAProxy and execute the tasks described above; hence, there is still room for further progress in this area.

Furthermore, the use of SMA is still a work-in-progress, as we are still exploring it and developing workflows to automate and orchestrate tasks. Its production infrastructure also needs to be scaled up as it does not have the number of runbook workers recommended by Microsoft - three, while we are using only one due to low workload, but more is expected in the near future; and a secondary database server is planned to provide redundancy for the data stored by SMA, in case of failure.

## BIBLIOGRAPHY

- [1] K. Morris. *Infrastructure as code: managing servers in the cloud*. O'Reilly Media, 2016. ISBN: 9781491924358. URL: <https://cds.cern.ch/record/2197640/export/hx?ln=en>.
- [2] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer. "What is DevOps? A Systematic Mapping Study on Definitions and Practices." In: *Proceedings of the Scientific Workshop Proceedings of XP2016 on - XP '16 Workshops* (2016), pp. 1–11. ISSN: 07421222. DOI: 10.1145/2962695.2962707. URL: <http://dl.acm.org/citation.cfm?doid=2962695.2962707>.
- [3] A. Dyck, R. Penners, and H. Lichter. *Towards Definitions for Release Engineering and DevOps*. 2015. URL: <https://www2.swc.rwth-aachen.de/docs/RELENG2015/DevOpsVsRelEng.pdf>.
- [4] L. E. Lwakatare, P. Kuvaja, and M. Oivo. "Dimensions of devOps." In: *Lecture Notes in Business Information Processing* 212 (2015), pp. 212–217. ISSN: 18651348. DOI: 10.1007/978-3-319-18612-2\_19.
- [5] F. Erich, C. Amrit, and M. Daneva. "Report: DevOps Literature Review." In: (2014). DOI: 10.13140/2.1.5125.1201. URL: <https://>.
- [6] P. Debois. "Devops: A Software Revolution in the Making?" In: *Cutter IT Journal* 24.8 (2011), pp. 1–41. ISSN: 15227383. URL: <https://www.cutter.com/article/devops-software-revolution-making-416511>.
- [7] R. Pressman. *Software Engineering: A Practitioner's Approach*. 7th ed. New York, NY, USA: McGraw-Hill, Inc., 2010. ISBN: 0073375977, 9780073375977.
- [8] Puppet. *Puppet*. URL: <https://puppet.com/product/how-puppet-works> (visited on 01/30/2017).
- [9] Chef. *Chef*. URL: <https://www.chef.io/chef/> (visited on 01/30/2017).
- [10] Microsoft. *Windows PowerShell Desired State Configuration Overview | Microsoft Docs*. URL: <https://docs.microsoft.com/en-us/powershell/dsc/overview> (visited on 06/20/2017).
- [11] L. Macvittie. *Automation versus Orchestration*. 2014. URL: <https://devops.com/automation-versus-orchestration/> (visited on 01/30/2017).

- [12] B. Saille. *Automation – Orchestrator Back to Basics – Use Cases*. 2013. URL: <https://blogs.technet.microsoft.com/privatecloud/2013/08/12/automation-orchestrator-back-to-basics-use-cases-spotlight-1-of-5/> (visited on 01/30/2017).
- [13] Microsoft. *Orchestrator*. URL: <https://technet.microsoft.com/en-us/system-center-docs/orch/orchestrator> (visited on 02/09/2017).
- [14] Microsoft. *Service Management Automation - Architecture*. 2016. URL: [https://technet.microsoft.com/en-us/library/dn469259\(v=sc.12\).aspx](https://technet.microsoft.com/en-us/library/dn469259(v=sc.12).aspx) (visited on 02/14/2017).
- [15] P Andrade, T Bell, J van Eldik, G McCance, B Panzer-Steindel, M Coelho dos Santos, S Traylen and, and U Schwickerath. “Review of CERN Data Centre Infrastructure.” In: *Journal of Physics: Conference Series* 396.4 (2012), p. 042002. DOI: 10.1088/1742-6596/396/4/042002. URL: <http://stacks.iop.org/1742-6596/396/i=4/a=042002?key=crossref.23670cb0bdd5ed06ad32f65944ef7af2>.
- [16] B Jones, G McCance, S Traylen, and N. B. Arias. “Scaling Agile Infrastructure to People.” In: *Journal of Physics: Conference Series* 664.2 (2015), p. 022026. ISSN: 1742-6588. DOI: 10.1088/1742-6596/664/2/022026. URL: <http://stacks.iop.org/1742-6596/664/i=2/a=022026?key=crossref.e23da1c5d6a28483958f807976be7189>.
- [17] OpenStack. *OpenStack*. URL: <https://www.openstack.org/> (visited on 02/09/2017).
- [18] CERN. *CERN - OpenStack*. URL: <https://clouddocs.web.cern.ch/clouddocs/overview/overview.html> (visited on 02/09/2017).
- [19] Microsoft. *Remote Desktop Services Overview*. URL: [https://technet.microsoft.com/en-us/library/cc725560\(v=ws.11\).aspx](https://technet.microsoft.com/en-us/library/cc725560(v=ws.11).aspx) (visited on 01/14/2017).
- [20] P. Mell and T. Grance. “The NIST definition of cloud computing.” In: *NIST Special Publication* 145 (2011), p. 7. ISSN: 00845612. DOI: 10.1136/emj.2010.096966. arXiv: 2305-0543. URL: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>.
- [21] B. Jones. “Using OpenStack and Puppet to deliver IaaS at CERN.” In: *Nuclear Electronics & Computing*. 2013. URL: [nec2013.jinr.ru/files/12/openstack-nec2013.pptx](http://nec2013.jinr.ru/files/12/openstack-nec2013.pptx).
- [22] OpenStack. *OpenStack Docs: Introduction to OpenStack*. URL: <https://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html> (visited on 09/07/2017).
- [23] CERN. *CERN Data Centre*. URL: <http://information-technology.web.cern.ch/about/computer-centre> (visited on 01/14/2017).
- [24] OpenStack. *OpenStack Docs: Block Storage*. URL: <https://docs.openstack.org/security-guide/block-storage.html> (visited on 09/09/2017).

- [25] OpenStack. *HypervisorSupportMatrix - OpenStack*. URL: <https://wiki.openstack.org/wiki/HypervisorSupportMatrix> (visited on 09/10/2017).
- [26] KVM. *KVM*. URL: [http://www.linux-kvm.org/page/Main\\_Page](http://www.linux-kvm.org/page/Main_Page) (visited on 02/09/2017).
- [27] Xen Project. *The Xen Project*. URL: <https://www.xenproject.org/> (visited on 09/10/2017).
- [28] Microsoft. *Hyper-V*. URL: [https://technet.microsoft.com/en-us/library/mt169373\(v=ws.11\).aspx](https://technet.microsoft.com/en-us/library/mt169373(v=ws.11).aspx) (visited on 02/09/2017).
- [29] VMWare. *ESXi | Bare Metal Hypervisor | VMware*. URL: <https://www.vmware.com/products/esxi-and-esx.html> (visited on 09/10/2017).
- [30] E. Bugnion, J. Nieh, D. Tsafirir, and D. Tsafirir. *Hardware and software support for virtualization*. Morgan & Claypool, 2017, p. 186. ISBN: 9781627056885. URL: <https://cds.cern.ch/record/2269601?ln=en>.
- [31] G. J. Popek and R. P. Goldberg. "Formal requirements for virtualizable third generation architectures." In: *Communications of the ACM* 17.7 (1974), pp. 412–421. ISSN: 01635980. DOI: 10.1145/957195.808061.
- [32] Red Hat. *Ceph storage*. URL: <https://ceph.com/ceph-storage/> (visited on 02/09/2017).
- [33] T Bell, B Bompastor, S Bukowiec, J Castro Leon, M. K. Denis, J van Eldik, M. F. Lobo, L. F. Alvarez, D. F. Rodriguez, A Marino, B Moreira, B Noel, T Oulevey, W Takase, A Wiebalck, and S Zilli. "Scaling the CERN OpenStack cloud." In: *Journal of Physics: Conference Series* 664.2 (2015), p. 022003. ISSN: 1742-6588. DOI: 10.1088/1742-6596/664/2/022003. URL: <http://stacks.iop.org/1742-6596/664/i=2/a=022003?key=crossref.4c93e737e9bf77d7d92793f48ecc6e7f>.
- [34] B. Moreira. "CERN Cloud Architecture." In: *Ops Midcycle - High Performance Computing with OpenStack - Manchester*. 2016.
- [35] B. Moreira. "Unveiling CERN Cloud Architecture." In: *Openstack Design Summit - Tokyo, 2015*. 2015.
- [36] B. Moreira. "Deep Dive into the CERN Cloud Infrastructure." In: *Openstack Design Summit - Hong Kong, 2013*. 2013. URL: <http://www.openstack.org/assets/presentation-media/Deep-Dive-into-the-CERN-Cloud-Infrastructure.pdf>.
- [37] Microsoft. *Remote Desktop Protocol (Windows)*. URL: [https://msdn.microsoft.com/en-us/library/aa383015\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa383015(v=vs.85).aspx) (visited on 01/14/2017).
- [38] Microsoft. *Remote Desktop Protocol (RDP)*. URL: <https://support.microsoft.com/en-us/help/186607/understanding-the-remote-desktop-protocol-rdp> (visited on 01/14/2017).

- [39] HAProxy. *HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer*. URL: <http://www.haproxy.org/#desc> (visited on 02/09/2017).
- [40] HAProxy. *HAProxy version 1.7.2 - Starter Guide*. URL: <http://cbonte.github.io/haproxy-dconv/1.7/intro.html> (visited on 02/09/2017).
- [41] M. Anicas. *HAProxy - An Introduction to HAProxy and Load Balancing Concepts | DigitalOcean*. 2014. URL: <https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts#types-of-load-balancing> (visited on 02/09/2017).
- [42] Keepalived. *Keepalived for Linux*. URL: <http://www.keepalived.org/> (visited on 02/09/2017).
- [43] J. Ellingwood. *Keepalived - How To Set Up Highly Available Web Servers with Keepalived and Floating IPs on Ubuntu 14.04 | DigitalOcean*. 2015. URL: <https://www.digitalocean.com/community/tutorials/how-to-set-up-highly-available-web-servers-with-keepalived-and-floating-ips-on-ubuntu-14-04> (visited on 02/11/2017).
- [44] M. Turnbull. *Open Source Windows service for reporting server load back to HAProxy (load balancer feedback agent)*. URL: <http://www.loadbalancer.org/blog/open-source-windows-service-for-reporting-server-load-back-to-haproxy-load-balancer-feedback-agent> (visited on 02/09/2017).
- [45] Loadbalancer.org. *HAProxy Feedback Agent*. URL: [https://github.com/loadbalancer-org/windows\\_feedback\\_agent](https://github.com/loadbalancer-org/windows_feedback_agent) (visited on 02/09/2017).
- [46] Microsoft. *Operations Manager Key Concepts*. URL: <https://technet.microsoft.com/library/hh230741.aspx>.
- [47] K. Greene. *Getting started with Microsoft system center operations manager : a beginner's guide to help you design, deploy and administer your Systems Center Operations Manager 2016 and 2012 R2 environments*. Packt Publ., 2016. ISBN: 9781785289743.
- [48] Microsoft. *Active Directory*. URL: <https://msdn.microsoft.com/en-us/library/bb742424.aspx#XSLTsection122121120120> (visited on 01/30/2017).
- [49] B. Desmond. *Active directory*. O'Reilly Media, 2013, p. 709. ISBN: 9781449320027.
- [50] S. Reimer. *Windows server 2008 Active Directory resource kit*. Microsoft Press, 2008, p. 827. ISBN: 0735625158.
- [51] Solveme.net. *What is Active Directory Directory Service in details? ADDS*. URL: <http://solveme.net/index.php/active-directory.html> (visited on 01/30/2017).
- [52] Microsoft. *JScript (ECMAScript3)*. URL: [https://msdn.microsoft.com/en-us/library/hbxc2t98\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/hbxc2t98(v=vs.85).aspx) (visited on 02/12/2017).
- [53] Microsoft. *What Is VBScript?* URL: <https://msdn.microsoft.com/en-us/library/1kw29xwf.aspx> (visited on 02/12/2017).



- [54] Microsoft. *Microsoft PowerShell*. URL: <https://msdn.microsoft.com/en-us/powershell/mt173057.aspx> (visited on 02/09/2017).
- [55] Microsoft. *Windows Management Instrumentation (Windows)*. URL: [https://msdn.microsoft.com/en-us/library/aa394582\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx) (visited on 01/30/2017).
- [56] M. Hester and S. Dutkiewicz. *Automating Microsoft Windows Server 2008 R2 with Windows Powershell 2.0*. Wiley Pub, 2011, p. 412. ISBN: 9781118013861.
- [57] Microsoft. *About WMI (Windows)*. URL: [https://msdn.microsoft.com/en-us/library/aa384642\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa384642(v=vs.85).aspx) (visited on 09/17/2017).
- [58] CERN. *CERN CMF - Help*. URL: <https://cmf.web.cern.ch/cmf/Help/?kbid=001001#1>.
- [59] Microsoft. *Systems Management Server (SMS) (Windows Embedded Standard 2009)*. URL: [https://msdn.microsoft.com/en-us/library/bb521519\(v=winembedded.51\).aspx](https://msdn.microsoft.com/en-us/library/bb521519(v=winembedded.51).aspx) (visited on 02/14/2017).
- [60] W. W. Rovce. "MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS." In: *Proceedings of IEEE WESCON* (1970). URL: <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>.
- [61] Tutorials Point. *SDLC Waterfall Model*. URL: [https://www.tutorialspoint.com/sdlc/sdlc\\_waterfall\\_model.htm](https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm) (visited on 07/01/2017).
- [62] J. Villa and K. Bauer. *Waterfall Model – TC1019 Fall 2016*. 2016. URL: <https://kenscourses.com/tc1019fall2016/syndicated/waterfall-model-4/#more-37588> (visited on 07/13/2017).
- [63] P. Poojary. *What is DevOps*. 2016. URL: [https://www.edureka.co/blog/what-is-devops/?utm\\_source=blog&utm\\_medium=left-menu&utm\\_campaign=devops-tutorial](https://www.edureka.co/blog/what-is-devops/?utm_source=blog&utm_medium=left-menu&utm_campaign=devops-tutorial) (visited on 06/01/2017).
- [64] V. Chaturvedi. *Introduction to DevOps*. 2016. URL: [https://www.edureka.co/blog/devops-tutorial?utm\\_source=blog&utm\\_medium=left-menu&utm\\_campaign=devops-tutorial](https://www.edureka.co/blog/devops-tutorial?utm_source=blog&utm_medium=left-menu&utm_campaign=devops-tutorial) (visited on 06/01/2017).
- [65] L. E. L. B, P. Kuvaja, and M. Oivo. "Relationship of DevOps to Agile, Lean and Continuous Deployment A Multivocal Literature Review Study." In: 10027 (2016), pp. 399–415. DOI: 10.1007/978-3-319-49094-6. URL: <http://link.springer.com/10.1007/978-3-319-49094-6>.
- [66] J. Allspaw and J. Robbins. *Web operations : keeping the data on time*. O'Reilly Media, 2010, p. 315. ISBN: 9781449377441. URL: <https://cds.cern.ch/record/1438357/export/hx?ln=en>.
- [67] P. Debois. "Agile infrastructure and operations: how infra-gile are you?" In: (2008). URL: <http://www.jedi.be/presentations/IEEE-Agile-Infrastructure.pdf>.

## BIBLIOGRAPHY

---

- [68] Wikipedia. *DevOps - Wikipedia*. URL: <https://en.wikipedia.org/wiki/DevOps> (visited on 01/30/2017).
- [69] Git. *Git*. URL: <https://git-scm.com/> (visited on 07/30/2017).
- [70] Apache. *Apache Subversion*. URL: <https://subversion.apache.org/> (visited on 07/30/2017).
- [71] Apache. *Apache Ant - Welcome*. URL: <http://ant.apache.org/> (visited on 07/30/2017).
- [72] Apache. *Maven – Welcome to Apache Maven*. URL: <https://maven.apache.org/> (visited on 07/30/2017).
- [73] Gradle. *Gradle Build Tool*. URL: <https://gradle.org/> (visited on 07/30/2017).
- [74] Selenium. *Selenium - Web Browser Automation*. URL: <http://www.seleniumhq.org/> (visited on 07/30/2017).
- [75] JUnit. *JUnit - About*. URL: <http://junit.org/junit4/> (visited on 07/30/2017).
- [76] S. Ambler. *Development Sandboxes: An Agile 'Best Practice'*. URL: <http://www.agiledata.org/essays/sandboxes.html> (visited on 07/30/2017).
- [77] P. Murray. *Traditional Development/Integration/Staging/Production Practice for Software Development | Disruptive Library Technology Jester*. 2006. URL: <http://dl.tj.org/article/software-development-practice/> (visited on 07/30/2017).
- [78] R. Ellison. "Software Testing Environments Best Practices." In: *Software Testing Magazine* (June 2016). URL: <http://www.softwaretestingmagazine.com/knowledge/software-testing-environments-best-practices/>.
- [79] Docker. *Docker - Build, Ship, and Run Any App, Anywhere*. URL: <https://www.docker.com/> (visited on 09/09/2017).
- [80] Jenkins. *Jenkins*. URL: <https://jenkins.io/> (visited on 07/30/2017).
- [81] Ansible. *Ansible is Simple IT Automation*. URL: <https://www.ansible.com/>.
- [82] Nagios. *Nagios - The Industry Standard In IT Infrastructure Monitoring*. URL: <https://www.nagios.org/> (visited on 07/30/2017).
- [83] New Relic. *Digital Performance Monitoring and Management | New Relic*. URL: <https://newrelic.com/> (visited on 07/30/2017).
- [84] Sensu. *Sensu | Full-stack monitoring for today's business*. URL: <https://sensuapp.org/> (visited on 07/30/2017).
- [85] R. Ahmed. *Ansible Tutorial | Ansible Playbooks And Adhoc Commands | Edureka*. 2016. URL: <https://www.edureka.co/blog/ansible-tutorial/> (visited on 06/01/2017).
- [86] A. Perilli. *Why Red Hat Acquired Ansible*. 2015. URL: <https://www.redhat.com/en/blog/why-red-hat-acquired-ansible> (visited on 08/27/2017).

- [87] YAML. *YAML Ain't Markup Language*. URL: <http://www.yaml.org/start.html> (visited on 08/27/2017).
- [88] M. Heap. *Ansible : from beginner to pro*. Apress, 2016. ISBN: 9781484216606.
- [89] Linux Information Project. *Daemon Definition*. 2005. URL: <http://www.linfo.org/daemon.html> (visited on 02/11/2017).
- [90] S. Krum, W. Van Hevelingen, B. Kero, J. Turnbull, and J. McCune. *Pro Puppet; 2nd ed*. The expert's voice in open source. New York, NY: Apress, 2013. URL: <https://cds.cern.ch/record/1665292>.
- [91] Puppet. *Getting started with classification — Documentation — Puppet*. 2017. URL: [https://docs.puppet.com/pe/latest/console\\_classes\\_groups\\_getting\\_started.html](https://docs.puppet.com/pe/latest/console_classes_groups_getting_started.html) (visited on 09/14/2017).
- [92] Puppet. *Grouping and classifying nodes — Documentation — Puppet*. 2017. URL: [https://docs.puppet.com/pe/2017.2/console\\_classes\\_groups.html](https://docs.puppet.com/pe/2017.2/console_classes_groups.html) (visited on 09/14/2017).
- [93] D. Lidral-Porter. "Node Classifier Fundamentals - Dan Lidral-Porter, Puppet Labs | Puppet." In: *Puppetconf 2014*. 2014. URL: <https://puppet.com/presentations/node-classifier-fundamentals-dan-lidral-porter-puppet-labs>.
- [94] Puppet. *Language: Resources — Documentation — Puppet*. 2017. URL: [https://docs.puppet.com/puppet/5.1/lang\\_resources.html#namenamevar](https://docs.puppet.com/puppet/5.1/lang_resources.html#namenamevar) (visited on 09/16/2017).
- [95] Puppet. *Language: Data types: Strings — Documentation — Puppet*. 2017. URL: [https://docs.puppet.com/puppet/latest/lang\\_data\\_string.html#double-quoted-strings](https://docs.puppet.com/puppet/latest/lang_data_string.html#double-quoted-strings) (visited on 09/17/2017).
- [96] Puppet. *Language: Conditional statements and expressions — Documentation — Puppet*. 2017. URL: [https://docs.puppet.com/puppet/5.2/lang\\_conditional.html](https://docs.puppet.com/puppet/5.2/lang_conditional.html) (visited on 09/17/2017).
- [97] CERN. *Puppet infrastructure - CERN*. URL: <http://configdocs.web.cern.ch/configdocs/overview/system.html> (visited on 02/14/2017).
- [98] Red Hat. "Ansible In Depth." In: (2016). URL: <https://www.ansible.com/ansible-in-depth-whitepaper>.
- [99] Microsoft. *Windows Remote Management (Windows)*. URL: [https://msdn.microsoft.com/en-us/library/aa384426\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa384426(v=vs.85).aspx) (visited on 09/21/2017).
- [100] DMTF. *WS-MAN*. 2013. URL: <https://www.dmtf.org/standards/wsman> (visited on 09/21/2017).
- [101] W. W. W. Consortium. *Simple Object Access Protocol*. 2000. URL: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508> (visited on 09/21/2017).

- [102] Red Hat. “The Benefits of Agentless Architecture.” In: (2016), pp. 1–6. URL: <https://www.ansible.com/benefits-of-agentless-architecture>.
- [103] J. McAllister. *Implementing DevOps with Ansible 2*. Packt Publishing, 2017. ISBN: 9781787126510. URL: <http://cds.cern.ch/record/2279889?ln=en>.
- [104] M. Mohaan and R. Raithatha. *Learning Ansible : use Ansible to configure your systems, deploy software, and orchestrate advanced IT tasks*. Packt Publ., 2014. ISBN: 9781783550647. URL: <http://cds.cern.ch/record/1985665?ln=en>.
- [105] Red Hat. *YAML Syntax — Ansible Documentation*. URL: <http://docs.ansible.com/ansible/latest/YAMLSyntax.html> (visited on 09/21/2017).
- [106] G. Garcia. *Ansible 2 - Cloud Modules*. 2015. URL: <http://slides.com/guidogarcia/ansible-2-0#/2> (visited on 09/21/2017).
- [107] Microsoft. *Get started with Desired State Configuration (DSC) for Linux | Microsoft Docs*. URL: <https://docs.microsoft.com/en-us/powershell/dsc/lnxgettingstarted> (visited on 09/22/2017).
- [108] Distributed Management Task Force. *CIM and MOF Tutorial*. 2017. URL: <https://www.dmtf.org/education/mof> (visited on 09/21/2017).
- [109] M. Gray. “An Overview of Windows PowerShell Desired State Configuration.” In: *TechEd Europe 2014*. 2014. URL: <http://video.ch9.ms/sessions/teched/eu/2014/CDP-B360.pptx>.
- [110] T. Turpijn. *Introducing PowerShell Desired State Configuration (DSC) | Building Clouds*. 2013. URL: <https://blogs.technet.microsoft.com/privatecloud/2013/08/30/introducing-powershell-desired-state-configuration-dsc/> (visited on 09/22/2017).
- [111] J. Pogram. *Learning PowerShell DSC*. Packt Publishing, 2015. ISBN: 9781783980703. URL: <http://cds.cern.ch/record/2113778>.
- [112] Microsoft. *Desired State Configuration Quick Start | Microsoft Docs*. URL: <https://docs.microsoft.com/en-us/powershell/dsc/quickstart> (visited on 09/22/2017).
- [113] Foreman. *Foreman*. 2017. URL: <https://www.theforeman.org/> (visited on 09/22/2017).
- [114] Sodabrew. *Puppet Dashboard*. 2017. URL: <https://github.com/sodabrew/puppet-dashboard> (visited on 09/22/2017).
- [115] Red Hat. *ARA*. 2016. URL: <https://ara.readthedocs.io/en/latest/index.html> (visited on 09/23/2017).
- [116] I. Pearson. *Tensor*. 2017. URL: <https://github.com/pearsonappeng/tensor/wiki> (visited on 09/23/2017).

- [117] Puppet. *Introduction to Puppet Enterprise Console* | Puppet. 2015. URL: <https://puppet.com/presentations/introduction-puppet-enterprise-console> (visited on 09/23/2017).
- [118] Red Hat. *Ansible Tower* | Ansible.com. URL: <https://www.ansible.com/tower> (visited on 09/23/2017).
- [119] R. Gaspar. *Contribution to puppet-wmi module*. 2017. URL: <https://github.com/ricardogaspar2/puppet-wmi>.
- [120] R. Gaspar. *Contribution to puppet-wmi module (Puppet Forge)*. 2017. URL: <https://forge.puppet.com/ricardogaspar2/wmi/readme>.
- [121] M. Stone. *souldo/wmi* · Puppet Forge. URL: <https://forge.puppet.com/souldo/wmi> (visited on 09/19/2017).
- [122] Puppet. *puppet/sslcertificate* · Puppet Forge. 2014. URL: <https://forge.puppet.com/puppet/sslcertificate> (visited on 09/19/2017).
- [123] Puppet. *puppet-sslcertificate* - GitHub. 2014. URL: <https://github.com/voxpupuli/puppet-sslcertificate> (visited on 09/19/2017).
- [124] R. Gaspar. *Contribution to puppet-sslcertificate module*. 2017. URL: <https://github.com/voxpupuli/puppet-sslcertificate/pull/46>.
- [125] Puppet. *Language: Resource collectors — Documentation — Puppet*. 2017. URL: [https://docs.puppet.com/puppet/5.2/lang\\_collectors.html](https://docs.puppet.com/puppet/5.2/lang_collectors.html) (visited on 09/20/2017).
- [126] B. Jones. *TeigiVault - Secrets for puppet*. 2013. URL: <https://twiki.cern.ch/twiki/bin/view/Main/TeigiVault> (visited on 09/19/2017).
- [127] Puppet. *Puppet Labs is now Puppet* | Puppet. 2016. URL: <https://puppet.com/puppet-labs-is-puppet> (visited on 09/18/2017).
- [128] R. Gaspar. *A Docker container to build Puppet types based on PowerShell DSC resources*. 2017. URL: <https://github.com/ricardogaspar2/puppet-dsc-build>.
- [129] G. Sarti. *Fundamental changes to the PowerShell DSC Module*. 2017. URL: [https://groups.google.com/forum/#!searchin/puppet-dev/For\\$20Comment\\$20-\\$20Fundamental\\$20changes\\$20to\\$20the\\$20PowerShell\\$20DSC\\$20Module%7Csort:relevance/puppet-dev/uQ642U3BwoQ/sQ1eRTpCCQAJ](https://groups.google.com/forum/#!searchin/puppet-dev/For$20Comment$20-$20Fundamental$20changes$20to$20the$20PowerShell$20DSC$20Module%7Csort:relevance/puppet-dev/uQ642U3BwoQ/sQ1eRTpCCQAJ) (visited on 09/18/2017).
- [130] Microsoft. *Microsoft - Runbook Concepts*. 2016. URL: [https://technet.microsoft.com/en-us/library/hh403820\(v=sc.12\).aspx](https://technet.microsoft.com/en-us/library/hh403820(v=sc.12).aspx) (visited on 02/12/2017).
- [131] Microsoft. *Contents of a Run Book*. 2012. URL: <https://technet.microsoft.com/library/Cc917702> (visited on 02/14/2017).
- [132] Microsoft. *Azure Automation Runbook Types* | Microsoft Docs. 2016. URL: <https://docs.microsoft.com/en-us/azure/automation/automation-runbook-types#graphical-runbooks> (visited on 02/12/2017).

## BIBLIOGRAPHY

---

- [133] Microsoft. *Orchestrator Architecture*. 2016. URL: [https://technet.microsoft.com/en-us/library/hh420377\(v=sc.12\).aspx](https://technet.microsoft.com/en-us/library/hh420377(v=sc.12).aspx) (visited on 02/12/2017).
- [134] Microsoft. *Microsoft System Center : designing orchestrator runbooks*. Microsoft Press, 2013. ISBN: 9780735682986.
- [135] M. Oliveira. *Microsoft System Center Orchestrator 2012 R2 essentials : design, implement, and improve your infrastructure administration with System Center Orchestrator 2012 R2's automation process*. Packt Publ., 2015. ISBN: 9781785287589.
- [136] S. Erskine, A. Baumgarten, and S. Beaumont. *Microsoft System Center 2012 Orchestrator cookbook*. Packt Pub, 2013. ISBN: 9781849688505.
- [137] Microsoft. *Microsoft Silverlight*. URL: <https://www.microsoft.com/silverlight/what-is-silverlight/> (visited on 02/12/2017).
- [138] Microsoft. *About Runbooks in Service Manager*. URL: [https://technet.microsoft.com/en-us/library/hh519715\(v=sc.12\).aspx](https://technet.microsoft.com/en-us/library/hh519715(v=sc.12).aspx) (visited on 02/12/2017).
- [139] Microsoft. *Windows Azure Pack overview*. URL: <https://www.microsoft.com/en-us/cloud-platform/windows-azure-pack> (visited on 02/12/2017).
- [140] Microsoft. *Simplify runbook authoring with global assets | Microsoft Docs*. 2017. URL: <https://docs.microsoft.com/en-us/system-center/sma/manage-global-assets?view=sc-sma-1711> (visited on 10/15/2017).
- [141] Microsoft. *Windows PowerShell Workflow Concepts*. 2016. URL: <https://docs.microsoft.com/en-us/system-center/sma/overview-powershell-workflows> (visited on 10/15/2017).
- [142] U. Windows, W. Foundation, and D. Chappell. "The Workflow Way." In: April (2009). URL: <https://msdn.microsoft.com/en-us/library/dd851337.aspx>.
- [143] M. Miller. *A Developer's Introduction to Windows Workflow Foundation (WF) in .NET 4*. 2010. URL: <https://msdn.microsoft.com/en-us/library/ee342461.aspx> (visited on 10/15/2017).
- [144] Microsoft. *Using Activities in Script Workflows*. 2015. URL: <https://technet.microsoft.com/en-us/library/jj574194.aspx> (visited on 10/15/2017).
- [145] Microsoft. *PowerShell Gallery*.
- [146] Microsoft. *System requirements for Service Management Automation | Microsoft Docs*. 2016. URL: <https://docs.microsoft.com/en-us/system-center/sma/system-requirements?view=sc-sma-1711> (visited on 10/15/2017).
- [147] I. Ahmad Khan. *ai-tools*. 2011. URL: <https://github.com/iahmad-khan/ai-tools>.
- [148] J. Levy. *PowerShell SSH module*. 2015. URL: <https://www.powershellgallery.com/packages/SSH/1.0.0> (visited on 10/15/2017).

- [149] S. Bukowiec, R. Gaspar, and T. S. Cern. “Windows Terminal Servers Orchestration 2016.” In: (2016). URL: <https://indico.cern.ch/event/505613/contributions/2227326/attachments/1346795/2037188/Poster-v3-31.pdf>.







## PUPPET TEST MANIFEST

---

```
1 class hg_playground::rchavesg::puppet_test {
2   # Change the value of a registry key
3   registry::value { 'Disable UAC':
4     key   => 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System',
5     value => 'EnableLUA',
6     data  => '0',
7     type  => 'dword'
8   }
9
10  # Stop a service - disable firewall
11  service { 'MpsSvc':
12    ensure => stopped,
13    enable => false
14  }
15
16  # Exec command - Set Powershell Execution Policy to unrestricted
17  exec { 'Set PowerShell execution policy unrestricted':
18    command => 'Set-ExecutionPolicy Unrestricted',
19    unless  => 'if ((Get-ExecutionPolicy -Scope LocalMachine) -eq "Unrestricted") { exit
20      ↪ 0 } else { exit 1 }',
21    provider => powershell
22  }
23
24  # File resource - Copy network directory to c:\temp (recursively)
25  file { 'Copy a directory':
26    ensure => directory,
27    path   => 'C:/temp',
28    recurse => true,
29    source => '//servershare/sharedir'
30  }
```

## ANNEX I. PUPPET TEST MANIFEST

---

```
30
31  # Install Remote Desktop Session Host server role
32  windowsfeature { 'RDS-RD-Server':
33    ensure => present,
34    restart => true
35  }
36 }
```

---



## ANSIBLE TEST PLAYBOOK

---

```
1  ---
2  - hosts: windows-ansible
3    tasks:
4      # Change the value of a registry key
5      - name: Disable UAC
6        win_regedit:
7          path: 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System'
8          name: 'EnableLUA'
9          data: 0
10         type: dword
11
12     # Stop a service - disable firewall
13     - name: Disable Firewall
14       win_service:
15         name: MpsSvc
16         state: stopped
17
18     # Execute a command - Set Powershell Execution Policy to unrestricted
19     - name: Set PowerShell Execution Policy
20       win_command: Set-ExecutionPolicy Unrestricted
21
22     # Copy network directory to c:\temp (recursively)
23     - name: Copy directory
24       win_copy:
25         src: \\servershare\sharedir
26         dest: c:\Temp
27
28     # Install Remote Desktop Session Host server role
29     - name: Install RDS
30       win_feature:
```

## ANNEX II. ANSIBLE TEST PLAYBOOK

---

```
31         name: RDS-RD-Server
32         state: present
33         restart: True
34     ...
```

---



## POWERSHELL DSC TEST CONFIGURATION SCRIPT

---

```
1 Configuration DSCtest
2 {
3     param
4     (
5         [string[]] $NodeName = 'localhost'
6     )
7
8     Import-DscResource -ModuleName PSDesiredStateConfiguration
9
10    Node $NodeName
11    {
12        # Change the value of a registry key
13        Registry DisableUAC
14        {
15            Ensure      = "Present"
16            Key          =
17                ↪ "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
18            ValueName   = "EnableLUA"
19            ValueData    = "0"
20            ValueType   = "Dword"
21        }
22
23        # Stop a service - disable firewall
24        Service DisableFirewall
25        {
26            Name        = "MpsSvc"
27            State        = "Stopped"
28        }
29    }
```

```
28
29     # Set Powershell Execution Policy to unrestricted
30     Script SetExecutionPolicy
31     {
32         SetScript = { Set-ExecutionPolicy Unrestricted }
33
34         TestScript = {
35             if ((Get-ExecutionPolicy -Scope LocalMachine) -eq "Unrestricted") {
36                 return $true
37             }else {
38                 return $false
39             }
40         }
41
42         GetScript = { @{ Result = (Get-ExecutionPolicy -Scope LocalMachine) } }
43     }
44
45     # Copy network directory to c:\temp (recursively)
46     File CopyDirectory
47     {
48         Ensure = "Present"
49         Type = "Directory"
50         Recurse = $true
51         SourcePath = "\\servershare\sharedir"
52         DestinationPath = "C:\temp"
53     }
54
55     # Install Remote Desktop Session Host server role
56     WindowsFeature RDSServer
57     {
58         Ensure = "Present"
59         Name = "RDS-RD-Server"
60     }
61 }
62
63 }
64
65 DSCtest
66 Start-DscConfiguration -Path .\DSCtest -Wait -Force -Verbose
```

---

## DSC CONFIGURATION SCRIPT TO SET GROUP POLICY RULES

---

```

1 Configuration LocalGP0
2 {
3     param([string[]] $NodeName = 'localhost')
4
5     Import-DSCResource -ModuleName PolicyFileEditor
6
7     Node $NodeName
8     {
9         cAdministrativeTemplateSetting RDSLicensing
10        {
11            KeyValueName = "SOFTWARE\Policies\Microsoft\Windows NT\Terminal
12            ↪ Services\LicenseServers"
13            PolicyType = "Machine"
14            Data = ("server.test.localgpo.dsc.com")
15            Ensure = "Present"
16            Type = "String"
17        }
18
19        cAdministrativeTemplateSetting RDSBrokerFarm
20        {
21            KeyValueName = "SOFTWARE\Policies\Microsoft\Windows NT\Terminal
22            ↪ Services\SessionDirectoryClusterName"
23            PolicyType = "Machine"
24            Data = ("server.test.localgpo.dsc.com")
25            Ensure = "Present"
26            Type = "String"
27        }
28    }
29 }

```

## ANNEX IV. DSC CONFIGURATION SCRIPT TO SET GROUP POLICY RULES

---

```
27     cAdministrativeTemplateSetting RDSBrokerName
28     {
29         KeyValueName = "SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
        ↳ Services\\SessionDirectoryLocation"
30         PolicyType = "Machine"
31         Data = ("server.test.localgpo.dsc.com")
32         Ensure = "Present"
33         Type = "String"
34     }
35
36     cAdministrativeTemplateSetting RDSLICENSEmode
37     {
38         KeyValueName = "SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
        ↳ Services\\LicensingMode"
39         PolicyType = "Machine"
40         Data = "2" # Per Device = 2; Per User = 4
41         Ensure = "Present"
42         Type = "DWord"
43     }
44 }
45 }
```

---





## MANIFEST TO SET GROUP POLICY RULES USING DSC RESOURCES

---

```

1 class hg_windows_dev::dsc_tests {
2   # test a custom resource
3   # Remote Desktop Services\Remote Desktop Session Host\Licensing\Use the specified
4   dsc_cadministrativetemplatesetting { 'test-dsc_cAdministrativeTemplateSetting License
    ↳ Server':
5     dsc_keyvaluenamename => 'SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↳ Services\\LicenseServers',
6     dsc_policytype      => 'Machine',
7     dsc_data             => 'server.test.localgpo.puppet-dsc.com',
8     dsc_ensure           => 'Present',
9     dsc_type             => 'String',
10  }
11
12  # Remote Desktop Services\Remote Desktop Session Host\RD Connection Broker\Configure RD
    ↳ Connection Broker farm Name
13  dsc_cadministrativetemplatesetting { 'test-dsc_cAdministrativeTemplateSetting RD
    ↳ Connection Broker farm Name':
14    dsc_keyvaluenamename => 'SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↳ Services\\SessionDirectoryClusterName',
15    dsc_policytype      => 'Machine',
16    dsc_data            => 'server.test.localgpo.puppet-dsc.com',
17    dsc_ensure          => 'Present',
18    dsc_type            => 'String',
19  }
20  }
21  # Remote Desktop Services\Remote Desktop Session Host\RD Connection Broker\Configure RD
    ↳ Connection Broker server Name
22  dsc_cadministrativetemplatesetting { 'test-dsc_cAdministrativeTemplateSetting RD
    ↳ Connection Broker server Name':

```

## ANNEX V. MANIFEST TO SET GROUP POLICY RULES USING DSC RESOURCES

---

```
23     dsc_keyvaluename => 'SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↪ Services\\SessionDirectoryLocation',
24     dsc_policytype   => 'Machine',
25     dsc_data         => 'server.test.localgpo.puppet-dsc.com',
26     dsc_ensure       => 'Present',
27     dsc_type         => 'String',
28
29   }
30
31   # Windows Components/Remote Desktop Services/Remote Desktop Session Host/Licensing
32   dsc_cadministrativetemplatesetting { 'test-dsc_cAdministrativeTemplateSetting dword':
33     dsc_keyvaluename => 'SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↪ Services\\LicensingMode',
34     dsc_policytype   => 'Machine',
35     dsc_data         => '2', # Per Device =2; Per User = 4
36     dsc_ensure       => 'Present',
37     dsc_type         => 'DWord',
38   }
39 }
```

---

## ANSIBLE PLAYBOOK TO SET GROUP POLICY RULES USING DSC RESOURCES

---

```

1  ---
2  - hosts: windows-ansible
3    tasks:
4      # GPO rules
5      # Remote Desktop Services\Remote Desktop Session Host\Licensing\Use the specified
6      - name: Set RDP Licensing server
7        win_dsc:
8          resource_name: cAdministrativeTemplateSetting
9          Ensure: "Present"
10         KeyValueName: "SOFTWARE\Policies\Microsoft\Windows NT\Terminal
11           ↳ Services\LicenseServers"
12         PolicyType: "Machine"
13         Data: "server.test.localgpo.puppet-dsc.com"
14         Type: "String"
15
16     # Remote Desktop Services\Remote Desktop Session Host\RD Connection Broker\Configure
17     ↳ RD Connection Broker farm Name
18     - name: Set RD Connection Broker farm Name
19       win_dsc:
20         resource_name: cAdministrativeTemplateSetting
21         Ensure: "Present"
22         KeyValueName: "SOFTWARE\Policies\Microsoft\Windows NT\Terminal
23           ↳ Services\SessionDirectoryClusterName"
24         PolicyType: "Machine"
25         Data: "server.test.localgpo.puppet-dsc.com"
26         Type: "String"
27
28     # Remote Desktop Services\Remote Desktop Session Host\RD Connection Broker\Configure
29     ↳ RD Connection Broker server Name

```

## ANNEX VI. ANSIBLE PLAYBOOK TO SET GROUP POLICY RULES USING DSC RESOURCES

---

```
26 - name: Set RD Connection Broker server Name
27   win_dsc:
28     resource_name: cAdministrativeTemplateSetting
29     Ensure: "Present"
30     KeyValueName: "SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↪ Services\\SessionDirectoryLocation"
31     PolicyType: "Machine"
32     Data: "server.test.localgpo.puppet-dsc.com"
33     Type: "String"
34
35   # Remote Desktop Services\\Remote Desktop Session Host\\RD Connection Broker\\Join RD
    ↪ Connection Broker
36 - name: "Join RD Connection Broker"
37   win_dsc:
38     resource_name: cAdministrativeTemplateSetting
39     Ensure: "Present"
40     KeyValueName: "SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↪ Services\\SessionDirectoryActive"
41     PolicyType: "Machine"
42     Data: "1" #Enable
43     Type: "DWord"
44
45   # Remote Desktop Services\\Remote Desktop Session Host\\Licensing
46 - name: "Set the Remote Desktop licensing mode"
47   win_dsc:
48     resource_name: cAdministrativeTemplateSetting
49     Ensure: "Present"
50     KeyValueName: "SOFTWARE\\Policies\\Microsoft\\Windows NT\\Terminal
    ↪ Services\\LicensingMode"
51     PolicyType: "Machine"
52     Data: "2" # Per Device =2; Per User = 4
53     Type: "DWord"
```

---

## ORIGINAL VERSION OF *puppet-wmi* MODULE

---

```

1  # == Defined Type: wmi
2  # This module is a defined type for manipulating WMI Objects with Puppet.
3  #
4  # === Parameters
5  # $wmi_namespace - The object's namespace.
6  # $wmi_class - The class name.
7  # $wmi_property - The property you wish to manage.
8  # $wmi_value - The value you wish for the property
9  # $wmi_method - The method to set the value, defaults to set${wmi_property}.
10 #
11 # === Examples
12 # wmi { 'Remote Desktop - Network Level Authentication' :
13 #   wmi_namespace => 'root/cimv2/terminalservices',
14 #   wmi_class      => 'Win32_TSGeneralSetting',
15 #   wmi_property   => 'UserAuthenticationRequired',
16 #   wmi_value      => 1,
17 # }
18 # wmi { 'Remote Desktop - Allow Connections' :
19 #   wmi_namespace => 'root/cimv2/terminalservices',
20 #   wmi_class      => 'Win32_TerminalServiceSetting',
21 #   wmi_property   => 'AllowTSConnections',
22 #   wmi_value      => 1
23 # }
24 #
25 # === Authors
26 # Matthew Stone <matt@souldo.net>
27 # === Copyright
28 # Copyright 2014 Matthew Stone, unless otherwise noted.
29 #
30 define wmi (

```

```
31  $wmi_namespace, $wmi_property, $wmi_class, $wmi_value, $wmi_method =  
    ↪ "Set${wmi_property}") {  
32  
33  $wmi_array = ["-Namespace ${wmi_namespace}",  
34    "-Class ${wmi_class}",]  
35  $wmi_data = join($wmi_array, ' ')  
36  $wmi_ps = "Get-WmiObject ${wmi_data}"  
37  $wmi_chk = "If ((\${wmiobject}.${wmi_property}) -like '${wmi_value}')"  
38  
39  exec { $name :  
40    command => "\${wmiobject}=${wmi_ps};\${wmiobject}.${wmi_method}(${wmi_value})",  
41    onlyif => "\${wmiobject}=${wmi_ps};${wmi_chk} { exit 1 }",  
42    provider => powershell,  
43  }  
44 }
```

---



## IMPROVED VERSION OF *puppet-wmi* MODULE

This version of the module can is available on GitHub [119].

---

```

1  # == Defined Type: wmi
2  # This module is a defined type for manipulating WMI Objects with Puppet.
3  #
4  # === Parameters
5  # $wmi_namespace - The object's namespace.
6  # $wmi_class - The class name.
7  # $wmi_property - The property you wish to manage.
8  # $wmi_value - The value you wish for the property
9  # $wmi_method - The method to set the value, defaults to set${wmi_property}.
10 #
11 # === Examples
12 # To use the wmi type, you must specify as least the namespace, class, property and value
13 # ↪ (as shown below).
14 # When a value for `wmi_method` is not provided it will change only the property
15 # ↪ specified.
16 # Otherwise, it will use the method provided.
17 # Be aware that you should always use the class method whenever it is available.
18 # That's the correct usage. Otherwise the property might not be changed properly.
19 #
20 # This example will set the property value directly without calling any method.
21 # Because there isn't any available.
22 # wmi { 'Change RDS RDP-TCP Environment Setting':
23 #   wmi_namespace => 'root/cimv2/terminalservices',
24 #   wmi_class      => 'Win32_TSEnvironmentSetting',
25 #   wmi_property   => 'InitialProgramPolicy',
26 #   wmi_value      => '2',
27 # }
28 #
29 # This example use specified methods.
```

```
28 # wmi { 'Remote Desktop - Allow Connections' :
29 #   wmi_namespace => 'root/cimv2/terminalservices',
30 #   wmi_class      => 'Win32_TerminalServiceSetting',
31 #   wmi_property   => 'AllowTSConnections',
32 #   wmi_value      => '1',
33 #   wmi_method     => 'SetAllowTSConnections',
34 # }
35 # === Authors
36 # Matthew Stone <matt@souldo.net>
37 # Changed by Ricardo Gaspar <ricardo.gaspar@cern.ch>
38 # === Copyright
39 # Copyright 2014 Matthew Stone, unless otherwise noted.
40 #
41 define wmi ($wmi_namespace, $wmi_property, $wmi_class, $wmi_value, $wmi_method = "") {
42   $wmi_array = ["-Namespace ${wmi_namespace}", "-Class ${wmi_class}",]
43   $wmi_data = join($wmi_array, ' ')
44   $wmi_ps = "Get-WmiObject ${wmi_data}"
45   $wmi_chk = "If ((\${wmiobject}.${wmi_property}) -like '${wmi_value}')"
46
47   if $wmi_method != "" {
48     $wmi_pscommand = "\${wmiobject}=${wmi_ps};\${wmiobject}.${wmi_method}('${wmi_value}')"
49   } else {
50     $wmi_pscommand = "${wmi_ps} | Set-WmiInstance -Arguments
51       ↪ @{${wmi_property}='${wmi_value}'}"
52   }
53
54   exec { $name:
55     command => "${wmi_pscommand}",
56     onlyif  => "\${wmiobject}=${wmi_ps};${wmi_chk} {exit 1} else {exit 0}",
57     provider => powershell,
58   }
```

---



## ORIGINAL VERSION OF *puppet-sslcertificate*

### MODULE MANIFEST

---

```
1 # Author:: Paul Stack (mailto:pstack@opentable.com)
2 # Copyright:: Copyright (c) 2013 OpenTable Inc
3 # License:: MIT
4
5 # == Define: sslcertificate
6 #
7 # This defined type will install SSL Certs on windows
8 #
9 # === Requirements/Dependencies
10 #
11 # Currently reequies the puppetlabs/stdlib module on the Puppet Forge in
12 # order to validate much of the the provided configuration.
13 #
14 # === Parameters
15 #
16 # [*password*]
17 # The password for the given certficate
18 #
19 # [*location*]
20 # The location to store intermediate certificates
21 #
22 # [*thumbprint*]
23 # The thumbprint used to verify the certficate
24 #
25 # [*store_dir*]
26 # The certificate store where the certificate will be installed to
27 #
28 # [*root_store*]
```

## ANNEX IX. ORIGINAL VERSION OF PUPPET-SSLCERTIFICATE MODULE MANIFEST

---

```
29 # The store location for the given certification store. Either LocalMachine or CurrentUser
30 #
31 # === Examples
32 #
33 # To install a certificate in the My directory of the LocalMachine root store:
34 #
35 # sslcertificate { "Install-PFX-Certificate" :
36 #   name      => 'mycert.pfx',
37 #   password  => 'password123',
38 #   location  => 'C:\',
39 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B'
40 # }
41 #
42 # To install a certificate in an alternative directory:
43 #
44 # sslcertificate { "Install-Intermediate-Certificate" :
45 #   name      => 'go_daddy_intermediate.p7b',
46 #   location  => 'C:\',
47 #   store_dir => 'CA',
48 #   root_store => 'LocalMachine',
49 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B'
50 # }
51 #
52 define sslcertificate($password, $location, $thumbprint, $root_store = 'LocalMachine',
53   ↳ $store_dir = 'My') {
54   validate_re($name, '^(.)+$', "Must pass name to ${module_name}[${title}]")
55   validate_re($location, '^(.)+$', "Must pass location to ${module_name}[${title}]")
56   validate_re($thumbprint, '^(.)+$', "Must pass a certificate thumbprint to
57     ↳ ${module_name}[${title}]")
58
59   ensure_resource('file', 'C:\temp', { ensure => directory })
60
61   file { "inspect-${name}-certificate.ps1" :
62     ensure => present,
63     path   => "C:\\temp\\inspect-${name}.ps1",
64     content => template('sslcertificate/inspect.ps1.erb'),
65     require => File['C:\temp'],
66   }
67
68   file { "import-${name}-certificate.ps1" :
69     ensure => present,
70     path   => "C:\\temp\\import-${name}.ps1",
71     content => template('sslcertificate/import.ps1.erb'),
72     require => File['C:\temp'],
73   }
74
75   exec { "Install-${name}-SSLCert":
76     provider => powershell,
77     command  => "c:\\temp\\import-${name}.ps1",
78     onlyif   => "c:\\temp\\inspect-${name}.ps1",
```

---

```
77     logoutoutput => true,  
78     require      => [ File["inspect-${name}-certificate.ps1"],  
    ↪ File["import-${name}-certificate.ps1"] ],  
79   }  
80 }
```

---



# A N N E X

## ORIGINAL VERSION OF INSPECT.PS1.ERB TEMPLATE

---

```

1 $pfx = new-object System.Security.Cryptography.X509Certificates.X509Certificate2
2
3 $certificate = gi <%= @location %>\<%= @name %>
4 switch -regex ($certificate.Extension.ToUpper()) {
5     ".CER|.DER|.P12" {
6         $pfx.import("<%= @location %>\<%= @name %>", "<%= @password
           ↳ %>", "Exportable,PersistKeySet")
7     }
8     ".CRT" {
9         $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\<%= @name %>"))
10    }
11    ".P7B|.SST" {
12        $pfx = new-object
           ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
13        $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\<%= @name %>"))
14    }
15    ".PFX" {
16        $pfx = new-object
           ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
17        $pfx.import("<%= @location %>\<%= @name %>", "<%= @password
           ↳ %>", "Exportable,PersistKeySet")
18    }
19 }
20
21
22 $installedCerts = @(Get-ChildItem -R cert:\<%= @root_store %>\<%= @store_dir %>)
23 $intermediateCerts = @(Get-ChildItem -R cert:\<%= @root_store %>\CA)
24

```

```
25 $installedCertCount = 0
26 $installedIntermediateCount = 0
27
28
29 if (($pfx -ne $null) -and ($installedCerts -ne $null) -and ($intermediateCerts -ne
↪ $null)) {
30     foreach($cert in $pfx)
31     {
32         if($cert.Thumbprint -ne "<%= @thumbprint %>") {
33             foreach ($intermediate in $intermediateCerts) {
34                 if($intermediate.Thumbprint -eq $cert.Thumbprint) {
35                     $installedIntermediateCount ++
36                 }
37             }
38         }
39         else {
40             foreach ($installedCert in $installedCerts) {
41                 if($installedCert.Thumbprint -eq $cert.Thumbprint) {
42                     $installedCertCount ++
43                 }
44             }
45         }
46     }
47
48     # When $pfx.Count is $null, $pfx is an instance of X509Certificate2, not
↪ X509Certificate2Collection, so
49     # ensure that only a single certificate has been installed.
50     if (($pfx.Count -eq $null) -and ($installedCertCount -eq 1) -and
↪ ($installedIntermediateCount -eq 0)) {
51         exit 1
52     }
53     elseif (($installedCertCount + $installedIntermediateCount) -eq $pfx.Count) {
54         exit 1
55     }
56 }
57
58 exit 0
```

---

## ORIGINAL VERSION OF IMPORT.PS1.ERB

### TEMPLATE

---

```

1 $pfx = new-object System.Security.Cryptography.X509Certificates.X509Certificate2
2
3 $cert = gi <%= @location %>/<%= @name %>
4
5 switch -regex ($cert.Extension.ToUpper()) {
6     ".CER|.DER|.P12" {
7         $pfx.import("<%= @location %>\\<%= @name %>", "<%= @password
            ↳ %>", "Exportable,PersistKeySet")
8     }
9     ".CRT" {
10         $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\\<%= @name %>"))
11     }
12     ".P7B|.SST" {
13         $pfx = new-object
            ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
14         $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\\<%= @name %>"))
15     }
16     ".PFX|.P12" {
17         $pfx = new-object
            ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
18         $pfx.import("<%= @location %>\\<%= @name %>", "<%= @password
            ↳ %>", "Exportable,PersistKeySet")
19     }
20 }
21
22 $store = new-object System.Security.Cryptography.X509Certificates.X509Store("<%=
            ↳ @store_dir %>", "<%= @root_store %>")
23 $store.open([System.Security.Cryptography.X509Certificates.OpenFlags]::ReadWrite)

```

```

24
25 $intermediatestore = new-object
    ↳ System.Security.Cryptography.X509Certificates.X509Store("CA", "<%= @root_store %>")
26 $intermediatestore.open([System.Security.Cryptography.X509Certificates.OpenFlags]::ReadWrite)
27
28 foreach($cert in $pfx) {
29     if($cert.Thumbprint -ne "<%= @thumbprint %>") {
30         $intermediatestore.Add($cert)
31     }
32     else {
33         $store.Add($cert)
34     }
35 }
36
37
38
39 $store.close()

```

---





## IMPROVED VERSION OF *puppet-sslcertificate*

### MODULE MANIFEST

---

```

1  # Author::    Paul Stack (mailto:pstack@opentable.com)
2  # Copyright:: Copyright (c) 2013 OpenTable Inc
3  # License::   MIT
4  # == Define: sslcertificate
5  #
6  # This defined type will install SSL Certs on windows
7  #
8  # === Requirements/Dependencies
9  #
10 # Currently reequires the puppetlabs/stdlib module on the Puppet Forge in
11 # order to validate much of the the provided configuration.
12 #
13 # === Parameters
14 #
15 # [*password*]
16 # The password for the given certificate
17 #
18 # [*location*]
19 # The location to store intermediate certificates.
20 # Do not end the string with any forward or backslash.
21 #
22 # [*thumbprint*]
23 # The thumbprint used to verify the certificate
24 #
25 # [*store_dir*]
26 # The certificate store where the certificate will be installed to
27 #
28 # [*root_store*]

```

## ANNEX XII. IMPROVED VERSION OF PUPPET-SSLCERTIFICATE MODULE MANIFEST

---

```
29 # The store location for the given certification store. Either LocalMachine or CurrentUser
30 #
31 # [*scripts_dir*]
32 # The directory where the scripts to verify and install the certificates will be stored.
33 # By default is C:\temp
34 #
35 # [*is_exportable*]
36 # Flag to set the key as exportable. true == exportable; false == not exportable.
37 # By default is set to true.
38 # === Examples
39 #
40 # To install a certificate in the My directory of the LocalMachine root store:
41 #
42 # sslcertificate { "Install-PFX-Certificate" :
43 #   name      => 'mycert.pfx',
44 #   password  => 'password123',
45 #   location  => 'C:',
46 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B'
47 # }
48 #
49 # To install a certificate in an alternative directory:
50 #
51 # sslcertificate { "Install-Intermediate-Certificate" :
52 #   name      => 'go_daddy_intermediate.p7b',
53 #   location  => 'C:',
54 #   store_dir => 'CA',
55 #   root_store => 'LocalMachine',
56 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B'
57 # }
58 #
59 # To install a certificate in the My directory of the LocalMachine root store
60 # using a different directory to store the scripts:
61 #
62 # sslcertificate { "Install-PFX-Certificate" :
63 #   name      => 'mycert.pfx',
64 #   password  => 'password123',
65 #   location  => 'C:',
66 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B',
67 #   scripts_dir => 'C:\scripts_dir'
68 # }
69 #
70 # To install a certificate in the My directory of the LocalMachine root store
71 # and set the key as not exportable:
72 #
73 # sslcertificate { "Install-PFX-Certificate" :
74 #   name      => 'mycert.pfx',
75 #   password  => 'password123',
76 #   location  => 'C:',
77 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B',
78 #   exportable => false
```

---

```

79 # }
80 #
81 define sslcertificate (
82     $password,
83     $location,
84     $thumbprint,
85     $root_store    = 'LocalMachine',
86     $store_dir     = 'My',
87     $scripts_dir  = 'C:\temp',
88     $exportable = true) {
89     validate_re($name, '^(.)+$', "Must pass name to ${module_name}[${title}]")
90     validate_re($location, '^(.)+$', "Must pass location to ${module_name}[${title}]")
91     validate_re($thumbprint, '^(.)+$', "Must pass a certificate thumbprint to
    ↪  ${module_name}[${title}]")
92
93     ensure_resource('file', $scripts_dir, {
94         ensure => directory
95     })
96
97     if $exportable {
98         $key_storage_flags = 'Exportable,PersistKeySet'
99     } else {
100         $key_storage_flags = 'PersistKeySet'
101     }
102
103     file { "inspect-${name}-certificate.ps1":
104         ensure => present,
105         path    => "${scripts_dir}\\inspect-${name}.ps1",
106         content => template('sslcertificate/inspect.ps1.erb'),
107         require => File[$scripts_dir],
108     }
109
110     file { "import-${name}-certificate.ps1":
111         ensure => present,
112         path    => "${scripts_dir}\\import-${name}.ps1",
113         content => template('sslcertificate/import.ps1.erb'),
114         require => File[$scripts_dir],
115     }
116
117     exec { "Install-${name}-SSLCert":
118         provider => powershell,
119         command  => "${scripts_dir}\\import-${name}.ps1",
120         onlyif   => "${scripts_dir}\\inspect-${name}.ps1",
121         logoutput => true,
122         require  => [File["inspect-${name}-certificate.ps1"],
    ↪  File["import-${name}-certificate.ps1"]],
123     }
124 }

```

---



## IMPROVED VERSION OF INSPECT.PS1.ERB

### TEMPLATE

---

```

1 $pfx = new-object System.Security.Cryptography.X509Certificates.X509Certificate2
2
3 $certificate = gi "<%= @location %>\<%= @name %>"
4 switch -regex ($certificate.Extension.ToUpper()) {
5     ".CER|.DER|.P12" {
6         $pfx.import("<%= @location %>\<%= @name %>","<%= @password %>","<%=
           ↳ @key_storage_flags %>")
7     }
8     ".CRT" {
9         $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\<%= @name %>"))
10    }
11    ".P7B|.SST" {
12        $pfx = new-object
           ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
13        $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\<%= @name %>"))
14    }
15    ".PFX" {
16        $pfx = new-object
           ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
17        $pfx.import("<%= @location %>\<%= @name %>","<%= @password %>","<%=
           ↳ @key_storage_flags %>")
18    }
19 }
20
21
22 $installedCerts = @(Get-ChildItem -R cert:\<%= @root_store %>\<%= @store_dir %>)
23 $intermediateCerts = @(Get-ChildItem -R cert:\<%= @root_store %>\CA)
24

```

```
25 $installedCertCount = 0
26 $installedIntermediateCount = 0
27
28
29 if (($pfx -ne $null) -and ($installedCerts -ne $null) -and ($intermediateCerts -ne
↪ $null)) {
30     foreach($cert in $pfx)
31     {
32         if($cert.Thumbprint -ne "<%= @thumbprint %>") {
33             foreach ($intermediate in $intermediateCerts) {
34                 if($intermediate.Thumbprint -eq $cert.Thumbprint) {
35                     $installedIntermediateCount ++
36                 }
37             }
38         }
39         else {
40             foreach ($installedCert in $installedCerts) {
41                 if($installedCert.Thumbprint -eq $cert.Thumbprint) {
42                     $installedCertCount ++
43                 }
44             }
45         }
46     }
47
48     # When $pfx.Count is $null, $pfx is an instance of X509Certificate2, not
↪ X509Certificate2Collection, so
49     # ensure that only a single certificate has been installed.
50     if (($pfx.Count -eq $null) -and ($installedCertCount -eq 1) -and
↪ ($installedIntermediateCount -eq 0)) {
51         exit 1
52     }
53     elseif (($installedCertCount + $installedIntermediateCount) -eq $pfx.Count) {
54         exit 1
55     }
56 }
57
58 exit 0
```

---

## IMPROVED VERSION OF IMPORT.PS1.ERB

### TEMPLATE

---

```

1 $pfx = new-object System.Security.Cryptography.X509Certificates.X509Certificate2
2
3 $cert = gi "<%= @location %>\<%= @name %>"
4
5 switch -regex ($cert.Extension.ToUpper()) {
6     ".CER|.DER|.P12" {
7         $pfx.import("<%= @location %>\<%= @name %>", "<%= @password %>", "<%=
            ↳ @key_storage_flags %>")
8     }
9     ".CRT" {
10         $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\<%= @name %>"))
11     }
12     ".P7B|.SST" {
13         $pfx = new-object
14         ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
15         $pfx.Import([System.IO.File]::ReadAllBytes("<%= @location %>\<%= @name %>"))
16     }
17     ".PFX" {
18         $pfx = new-object
19         ↳ System.Security.Cryptography.X509Certificates.X509Certificate2Collection
20         $pfx.import("<%= @location %>\<%= @name %>", "<%= @password %>", "<%=
            ↳ @key_storage_flags %>")
21     }
22 }
23
24 $store = new-object System.Security.Cryptography.X509Certificates.X509Store("<%=
    ↳ @store_dir %>", "<%= @root_store %>")
25 $store.open([System.Security.Cryptography.X509Certificates.OpenFlags]::ReadWrite)

```

```
24
25 $intermediatestore = new-object
    ↳ System.Security.Cryptography.X509Certificates.X509Store("CA", "<%= @root_store %>")
26 $intermediatestore.open([System.Security.Cryptography.X509Certificates.OpenFlags]::ReadWrite)
27
28 foreach($cert in $pfx) {
29     if($cert.Thumbprint -ne "<%= @thumbprint %>") {
30         $intermediatestore.Add($cert)
31     }
32     else {
33         $store.Add($cert)
34     }
35 }
36
37
38
39 $store.close()
```

---





## *cernsslcertificate* MODULE MANIFEST

---

```

1  # This is a wrapper for the sslcertificate module that allows to provide
2  # a password using the TEIGI module and thus avoiding the use of clear text passwords.
3  #
4  # This defined type will install SSL Certificates on windows
5  #
6  # === Requirements/Dependencies
7  # Same as sslcertificate
8  #
9  # === Parameters
10 #
11 # [*teigi_key*]
12 # TEIGI key that points to the password.
13 #
14 # [*location*]
15 # The location where the file certificate is.
16 # Do not end the string with any forward or backslash.
17 #
18 # [*thumbprint*]
19 # The thumbprint used to verify the certificate
20 #
21 # [*store_dir*]
22 # The certificate store where the certificate will be installed to
23 #
24 # [*root_store*]
25 # The store location for the given certification store. Either LocalMachine or CurrentUser
26 #
27 # [*scripts_dir*]
28 # The directory where the scripts to verify and install the certificates will be stored.
29 # By default is C:\temp
30 #

```

```
31 # [*exportable*]
32 # Flag to set the key as exportable. true == exportable; false == not exportable.
33 # By default is set to false.
34 # === Examples
35 #
36 # To install a certificate in the My directory of the LocalMachine root store:
37 #
38 # cernsslcertificate { "Install-PFX-Certificate" :
39 #   name      => 'mycert.pfx',
40 #   teigi_key  => 'passwordkey123',
41 #   location  => 'C:',
42 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B'
43 # }
44 #
45 # To install a certificate in an alterntative direcotory:
46 #
47 # cernsslcertificate { "Install-Intermediate-Certificate" :
48 #   name      => 'go_daddy_intermediate.p7b',
49 #   location  => 'C:',
50 #   store_dir => 'CA',
51 #   root_store => 'LocalMachine',
52 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B'
53 # }
54 #
55 # To install a certificate in the My directory of the LocalMachine root store
56 # using a different directory to store the scripts:
57 #
58 # cernsslcertificate { "Install-PFX-Certificate" :
59 #   name      => 'mycert.pfx',
60 #   teigi_key  => 'passwordkey123',
61 #   location  => 'C:',
62 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B',
63 #   scripts_dir => 'C:\scripts_dir'
64 # }
65 #
66 # To install a certificate in the My directory of the LocalMachine root store
67 # and set the key as exportable:
68 #
69 # cernsslcertificate { "Install-PFX-Certificate" :
70 #   name      => 'mycert.pfx',
71 #   teigi_key  => 'passwordkey123',
72 #   location  => 'C:',
73 #   thumbprint => '07E5C1AF7F5223CB975CC29B5455642F5570798B',
74 #   exportable => true
75 # }
76
77 define cernsslcertificate (
78   $teigi_key,
79   $location,
80   $thumbprint,
```

---

```

81  $root_store = 'LocalMachine',
82  $store_dir  = 'My',
83  $scripts_dir = 'C:\ProgramData\PuppetLabs\puppet\cache\tdag\cert-scripts',
84  $exportable = false) {
85
86  $cert_filename = $name
87
88  sslcertificate { "Install-S{cert_filename}":
89      name          => $cert_filename,
90      password      => "%TEIGI__${teigi_key}__%",
91      location      => $location,
92      thumbprint    => $thumbprint,
93      scripts_dir   => $scripts_dir,
94      exportable    => $exportable,
95  }
96
97  $inspect_script_name = "inspect-${cert_filename}-certificate.ps1"
98  $import_script_name = "import-${cert_filename}-certificate.ps1"
99
100  # Override scripts location and names defined by sslcertificate module.
101  # This will produce script files without the password in them.
102  File <| title == "${inspect_script_name}" |> {
103      path => "${scripts_dir}\\${inspect_script_name}.withoutteigi",
104  }
105
106  File <| title == "${import_script_name}" |> {
107      path => "${scripts_dir}\\${import_script_name}.withoutteigi",
108  }
109
110  # Override exec commands to use the correct script file names
111  Exec <| title == "Install-${cert_filename}-SSLCert" |> {
112      command => "${scripts_dir}\\${import_script_name}",
113      onlyif  => "${scripts_dir}\\${inspect_script_name}",
114  }
115
116  # These will be executed locally in each node. So the replacement and generation of
↪ scripts that include the password will be done
117  # by the host.
118  teigi::secret::sub_file { "${scripts_dir}\\${inspect_script_name}":
119      teigi_keys => [$teigi_key],
120      source     =>
↪ "${scripts_dir}\\inspect-${cert_filename}-certificate.ps1.withoutteigi",
121      before     => Exec["Install-${cert_filename}-SSLCert"],
122      require    => File[$inspect_script_name],
123      owner      => 'S-1-5-18',
124      group      => 'S-1-5-32-544',
125      mode       => '0700'
126  }
127
128  teigi::secret::sub_file { "${scripts_dir}\\${import_script_name}":

```

```
129     teigi_keys => [$teigi_key],
130     source     => "${scripts_dir}\\${import_script_name}.withoutteigi",
131     before     => Exec["Install-${cert_filename}-SSLCert"],
132     require    => File[$import_script_name],
133     owner      => 'S-1-5-18',
134     group      => 'S-1-5-32-544',
135     mode       => '0700'
136 }
137
138 }
```

---

## WINDOWS PROVIDER FOR TEIGI\_SUBFILE

---

```

1  Puppet::Type.type(:teigi_sub_file).provide(:windows) do
2    desc 'Replace strings in a file with secrets from teigi'
3    #####
4    ##### Code copied & adapted from the windows provider for file resource   ###
5    ##### located at /usr/share/ruby/vendor_ruby/puppet/provider/file/windows.rb #####
6    #####
7    confine operatingsystem: :windows
8    defaultfor osfamily: :windows
9    has_feature :manages_symlinks if Puppet.features.manages_symlinks?
10   include Puppet::Util::Warnings
11   if Puppet.features.microsoft_windows?
12     require 'puppet/util/windows'
13     include Puppet::Util::Windows::Security
14   end
15   # Determine if the account is valid, and if so, return the UID
16   def name2id(value)
17     Puppet::Util::Windows::SID.name_to_sid(value)
18   end
19   # If it's a valid SID, get the name. Otherwise, it's already a name,
20   # so just return it.
21   def id2name(id)
22     if Puppet::Util::Windows::SID.valid_sid?(id)
23       Puppet::Util::Windows::SID.sid_to_name(id)
24     else
25       id
26     end
27   end
28   # We use users and groups interchangeably, so use the same methods for both
29   # (the type expects different methods, so we have to oblige).
30   alias_method :uid2name, :id2name

```

```
31     alias_method :gid2name, :id2name
32     alias_method :name2gid, :name2id
33     alias_method :name2uid, :name2id
34     def owner
35         return :absent unless File.stat(resource[:destfile])
36         get_owner(resource[:destfile])
37     end
38     def owner=(should)
39         set_owner(should, resource[:destfile])
40     rescue => detail
41         raise Puppet::Error, "Failed to set owner to '#{should}': #{detail}",
42             ↳ detail.backtrace
43     end
44     def group
45         return :absent unless File.stat(resource[:destfile])
46         get_group(resource[:destfile])
47     end
48     def group=(should)
49         set_group(should, resource[:destfile])
50     rescue => detail
51         raise Puppet::Error, "Failed to set group to '#{should}': #{detail}",
52             ↳ detail.backtrace
53     end
54     def mode
55         if File.stat(resource[:destfile])
56             mode = get_mode(resource[:destfile])
57             mode ? mode.to_s(8) : :absent
58         else
59             :absent
60         end
61     end
62     def mode=(value)
63         begin
64             set_mode(value.to_i(8), resource[:destfile])
65         rescue => detail
66             error = Puppet::Error.new("failed to set mode #{mode} on
67                 ↳ #{resource[:destfile]}: #{detail.message}")
68             error.set_backtrace detail.backtrace
69             raise error
70         end
71         :file_changed
72     end
73     def create
74         target_contents = self.new_contents
75         if File.exist?(resource[:destfile])
76             target_current_contents = IO.read(resource[:destfile])
77             if target_current_contents == target_contents
```

---

```

78         set_perm
79         return true
80     end
81 end
82 File.open(resource[:destfile], File::WRONLY | File::TRUNC | File::CREAT) { |file|
83     ↪ file.write(target_contents) }
84 set_perm
85 end
86 def destroy
87     FileUtils.rm resource[:destfile] if File.exist? resource[:destfile]
88 end
89 def exists?
90     return false unless File.exist?(resource[:destfile])
91     target_contents = self.new_contents
92     current_contents = IO.read(resource[:destfile])
93     return false unless target_contents == current_contents
94     dest_stat = File.stat(resource[:destfile])
95     unless dest_stat.mode.to_s(8)[-4..-1] =~ /^0?#{resource[:mode]}$/
96         return false
97     end
98     return false if uid2name(dest_stat.uid) != resource[:owner]
99     return false if gid2name(dest_stat.gid) != resource[:group]
100     true
101 end
102 def new_contents
103     secret_contents = get_contents
104     unless secret_contents['errors'].empty?
105         raise secret_contents['errors'].join("\n")
106     end
107     source_contents = self.source_contents
108     keys = []
109     if resource[:teigi_keys].is_a?(Array)
110         keys = resource[:teigi_keys]
111     elsif resource[:teigi_keys].is_a?(String)
112         keys.push(resource[:teigi_keys])
113     else
114         raise("Unknown type #{resource[:teigi_keys].class} for teigi_keys")
115     end
116     keys.each do |k|
117         tbreplaced = resource[:string].sub('keyname', k)
118         @my_value = secret_contents['content'][k]
119         if resource[:encode] && (resource[:encode] != :false)
120             @my_value = ERB::Util.url_encode(@my_value)
121         end
122         source_contents = source_contents.gsub(tbreplaced, @my_value)
123     end
124     source_contents
125 end
126 def source_contents
127     return @source_contents if defined?(@source_contents)

```

```
127     @source_contents = IO.read(resource[:sourcefile])
128     @source_contents
129   end
130   def get_contents
131     return @results if defined?(@results)
132     dirpath = "#{Facter[:puppet_vardir].value}/tbag" # FIXME
133     @results = {}
134     @results['content'] = {}
135     @results['errors'] = {}
136     keys = []
137     if resource[:teigi_keys].is_a?(Array)
138       keys = resource[:teigi_keys]
139     elsif resource[:teigi_keys].is_a?(String)
140       keys.push(resource[:teigi_keys])
141     else
142       raise("Unknown type #{resource[:teigi_keys].class} for teigi_keys")
143     end
144     keys.each do |k|
145       k_path = [dirpath, k].join('/')
146       unless File.file?(k_path)
147         @results['errors'][k] = "teigisecret[\"#{k}\"] does not exist"
148         next
149       end
150       begin
151         @results['content'][k] = File.open(k_path, &:readline).chomp
152       rescue EOFError
153         # empty secret
154         @results['content'][k] = ''
155       end
156     end
157     @results
158   end
159   def set_perm
160     uid = name2uid(resource[:owner])
161     gid = name2gid(resource[:group])
162     self.owner = uid
163     self.group = gid
164     # Do not set mode if none is specified
165     if resource[:mode]
166       self.mode = resource[:mode]
167     end
168   end
169 end
```

---



## IMPROVED TEIGI\_SUBFILE MANIFEST TO CONTEMPLATE WINDOWS SYSTEMS

---

```

1  define teigi::secret::sub_file (
2      Optional[Array[String[1],1]] $teigi_keys = undef,
3      String $path=$title,
4      Optional[String] $template = undef,
5      Optional[String] $content = undef,
6      Optional[String] $source = undef,
7      Optional[String] $owner = undef,
8      Optional[String] $group = undef,
9      Optional[String] $mode = undef,
10     String $string = '%TEIGI__keyname__%',
11     Optional[Boolean] $encode = undef,
12 ) {
13
14     if $template and $source {
15         fail('$source and $template cannot be both set to teigi::sub::file')
16     } elseif $template and $content {
17         fail('$content and $template cannot be both set to teigi::sub::file')
18     } elseif $source and $content {
19         fail('$source and $content cannot be both set to teigi::sub::file')
20     } elseif $template {
21         $_mycontent = template($template)
22         $_mysource = undef
23     } elseif $source {
24         $_mycontent = undef
25         $_mysource = $source
26     } elseif $content {
27         $_mycontent = $content
28         $_mysource = undef

```

## ANNEX XVII. IMPROVED TEIGI\_SUBFILE MANIFEST TO CONTEMPLATE WINDOWS SYSTEMS

---

```
29   } else {
30     fail('On of $source, $template or $content must be set to teigi::sub::file')
31   }
32
33   # If provided, use the values passed as arguments. Otherwise, use the default ones
34   if $owner {
35     $_owner = $owner
36   } else {
37     $_owner = $::kernel ? {
38       'windows' => 'S-1-5-18', # SYSTEM account
39       default   => 'root',
40     }
41   }
42
43   if $group {
44     $_group = $group
45   } else {
46     $_group = $::kernel ? {
47       'windows' => 'S-1-5-32-544', # Administrators group account
48       default   => 'root',
49     }
50   }
51
52   if $mode {
53     $_mode = $mode
54   } else {
55     $_mode = $::kernel ? {
56       'windows' => '0700',
57       default   => '0400',
58     }
59   }
60
61   if $teigi_keys {
62     include ::teigi::tbag
63
64     $intermediate_file = md5($path)
65
66     $_intermediatemode = $::kernel ? {
67       'windows' => '0700',
68       default   => '0400',
69     }
70
71     file{"${teigi::tbag::template_path}/${intermediate_file}":
72       ensure => file,
73       owner  => $_owner,
74       group  => $_group,
75       mode   => $_intermediatemode,
76       content => $_mycontent,
77       source => $_mysource,
78   }
```

---

```

79
80 ensure_resource('teigisecret', $teigi_keys, {'path' => $teigi::tbag::path, 'url' =>
    ↳ $teigi::tbag::url, 'urlargs' => $teigi::tbag::urlargs, 'metapath' =>
    ↳ $teigi::tbag::metapath, 'require' => "File[${teigi::tbag::path}]")
81
82 file{$path:
83     ensure => present,
84     owner  => $_owner,
85     group  => $_group,
86     mode   => $_mode,
87 }
88
89 $_provider = $::kernel ? {
90     'windows' => 'windows',
91     default   => 'posix'
92 }
93
94 teigi_sub_file{$title:
95     teigi_keys => $teigi_keys,
96     provider   => $_provider,
97     string     => $string,
98     owner      => $_owner,
99     group      => $_group,
100    mode        => $_mode,
101    sourcefile  => "${teigi::tbag::template_path}/${intermediate_file}",
102    destfile    => $path,
103    require     =>
        ↳ [File["${teigi::tbag::template_path}/${intermediate_file}", $path], Teigisecret[$teigi_keys]],
104    encode      => $encode,
105 }
106 } else {
107
108     file{$path:
109         ensure => present,
110         owner  => $_owner,
111         group  => $_group,
112         mode   => $_mode,
113         content => $_mycontent,
114         source  => $_mysource,
115     }
116 }
117 }

```

---



# ANNEX XVIII

## PUPPET MANIFEST TO CONFIGURE REMOTE DESKTOP SERVERS OF CERN's WTS

---

```

1 class hg_windows_infrastructure::ts {
2   if $::operatingsystem == 'windows' {
3     # Setting Powershell Execution Policy to unrestricted
4     exec { 'Set PowerShell execution policy unrestricted':
5       command => 'Set-ExecutionPolicy Unrestricted',
6       unless  => 'if ((Get-ExecutionPolicy -Scope LocalMachine) -eq "Unrestricted") {
7         ↪ exit 0 } else { exit 1 }',
8       provider => powershell
9     }
10    #####
11    # Setting registry key changes #
12    #####
13    registry::value { 'Disable UAC':
14      key   => 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System',
15      value => 'EnableLUA',
16      data  => '0',
17      type  => 'dword'
18    }
19    registry::value { 'Set Updates':
20      key   => 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WindowsUpdate\Auto
21        ↪ Update',
22      value => 'AUOptions',
23      data  => '2',
24      type  => 'dword'
25    }
26    registry::value { 'Set Updates2':
27      key   => 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WindowsUpdate\Auto
28        ↪ Update',
29      value => 'IncludeRecommendedUpdates',

```

## ANNEX XVIII. PUPPET MANIFEST TO CONFIGURE REMOTE DESKTOP SERVERS OF CERN'S WTS

---

```
27     data => '1',
28     type => 'dword'
29   }
30   registry::value { 'Set IE Enhanced Security Configuration for Admins':
31     key   => 'HKLM\SOFTWARE\Microsoft\Active Setup\Installed
32       ↳ Components\{A509B1A7-37EF-4b3f-8CFC-4F3A74704073}',
33     value => 'IsInstalled',
34     data  => '0',
35     type  => 'dword'
36   }
37   registry::value { 'Set IE Enhanced Security Configuration for Users':
38     key   => 'HKLM\SOFTWARE\Microsoft\Active Setup\Installed
39       ↳ Components\{A509B1A8-37EF-4b3f-8CFC-4F3A74704073}',
40     value => 'IsInstalled',
41     data  => '0',
42     type  => 'dword'
43   }
44   registry::value { 'Disable Customer Experience Improvement Program':
45     key   => 'HKLM\SOFTWARE\Microsoft\SQMClient\Windows',
46     value => 'CEIPEnable',
47     data  => '0',
48     type  => 'dword'
49   }
50   registry::value { 'Disable Initial Configuration Tasks':
51     key   => 'HKLM\Software\Microsoft\ServerManager\Oobe',
52     value => 'DoNotOpenInitialConfigurationTasksAtLogon',
53     data  => '1',
54     type  => 'dword'
55   }
56   registry::value { 'Change Kerberos token size':
57     key   => 'HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters',
58     value => 'MaxTokenSize',
59     data  => '0xffff',
60     type  => 'dword'
61   }
62   # Set RDP
63   registry::value { 'Set RDP':
64     key   => 'HKLM\SYSTEM\CurrentControlSet\Control\Terminal
65       ↳ Server\WinStations\RDP-Tcp',
66     value => 'UserAuthentication',
67     data  => '0',
68     type  => 'dword'
69   }
70   # Install Remote Desktop Session Host role
71   windowsfeature { 'RDS-RD-Server':
72     ensure => present,
73     restart => true
74   }
75   wmi { 'Change RDS RDP-TCP Environment Setting':
76     wmi_namespace => 'root/cimv2/terminalservices',
```

---

```

74     wmi_class      => 'Win32_TSEnvironmentSetting',
75     wmi_property   => 'InitialProgramPolicy',
76     wmi_value      => 2,
77 }
78 # Setting the page file
79 $pagefile_initialsize = 2048
80 $pagefile_maximumpsize = 4096
81 exec { 'Set-pagefile':
82     command => "Get-CimInstance -ClassName Win32_ComputerSystem | Set-CimInstance
      ↪ -Property @{ AutomaticManagedPageFile = \$false }; Get-CimInstance -ClassName
      ↪ Win32_PageFileSetting | Set-CimInstance -Property @{InitialSize =
      ↪ $pagefile_initialsize; MaximumSize = $pagefile_maximumpsize}",
83     onlyif   => "\$AutoManaged = (Get-CimInstance -ClassName
      ↪ Win32_ComputerSystem).AutomaticManagedPagefile; \$PageFile = Get-CimInstance
      ↪ -ClassName Win32_PageFileSetting; if(\$AutoManaged -or \$PageFile.InitialSize
      ↪ -ne $pagefile_initialsize -or \$PageFile.MaximumSize -ne
      ↪ $pagefile_maximumpsize){exit 0} else {exit 1}",
84     provider => powershell
85 }
86 # Reboot after setting page file parameters
87 reboot { 'reboot after pagefile':
88     apply      => finished,
89     subscribe  => Exec['Set-pagefile'],
90     message    => 'Puppet changed the page file settings. This computer will be rebooted
      ↪ now.'
91 }
92 # Change start type of Software Protection Service to Manual
93 if ($::operatingsystemrelease == '2008 R2') {
94     # Only for 2008 R2, 2012 R2 gives an error:Cannot enable sppsvc for manual start,
      ↪ error was: Input/output error -
95     # ChangeServiceConfig: Access is denied.
96     service { 'sppsvc': enable => manual }
97 }
98 # Enable Powershell Remoting
99 exec { 'Enable PS Remoting':
100     command => 'Enable-PSRemoting -Force -Confirm:$false',
101     unless  => '$PSPermissionStr = (Get-PSSessionConfiguration).Permission |
      ↪ Select-String "AccessDenied"; if ($PSPermissionStr.Length -eq 0) {exit 0} else
      ↪ {exit 1}',
102     provider => powershell
103 }
104 # Disable IPV6 using windows_disable_ipv6 module
105 class { 'windows_disable_ipv6':
106     ipv6_disable => true,
107     ipv6_reboot  => false
108 }
109 # Disable TCP Chimney and NetDMA
110 exec { 'Disable TCP Chimney':
111     command => 'netsh int tcp set global chimney=disabled',

```

## ANNEX XVIII. PUPPET MANIFEST TO CONFIGURE REMOTE DESKTOP SERVERS OF CERN'S WTS

---

```
112     unless => '$ChimneyState = netsh int tcp show global | Select-String "Chimney
113         ↳ Offload State"; if($ChimneyState -like "*disabled*") {exit 0} else { exit 1}',
114     provider => powershell
115 }
116 registry::value { 'Disable NetDMA':
117     key    => 'HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters',
118     value  => 'EnableTCPA',
119     data   => '0',
120     type   => 'dword'
121 }
122 # Copy NICETasks folder (3 steps)
123 $nicetasks_source_dir = '//OFFUSCATEDPATH/TerminalServices/configuration/NICETasks'
124 $nicetasks_target_dir = 'C:/NICETasks'
125
126 #1 CreateMake sure folder a folder exists
127 file { $nicetasks_target_dir:
128     ensure =>    directory
129 }
130 #2 set permissions
131 acl { "set-permissions-${nicetasks_target_dir}":
132     target          => $nicetasks_target_dir,
133     purge           => true,
134     permissions     => [
135         { identity => 'SYSTEM', rights => ["full"], child_types => 'all', affects =>
136             ↳ 'all' },
137         { identity => 'Administrators', rights => ["full"], child_types => 'all', affects
138             ↳ => 'all' },
139     ],
140     owner            => 'Administrators',
141     inherit_parent_permissions => false,
142     require          =>    File[$nicetasks_target_dir],
143 }
144 #3 sync folder contents
145 exec { 'Sync NICETasks':
146     command => "\\\\OFFUSCATEDPATH\\TerminalServices\\scripts\\Sync-Files.ps1
147         ↳ ${nicetasks_source_dir} ${nicetasks_target_dir}",
148     provider => powershell,
149     require  =>    Acl["set-permissions-${nicetasks_target_dir}"]
150 }
151
152 $cert_thumbprint = 'OFFUSCATEDTHUMBPRINT'
153
154 # Import the some certificate
155 cernsslcertificate { 'INSTALL-CERTIFICATE':
156     name          => 'OFFUSCATEDPCERTIFICATE.pfx',
157     teigi_key     => 'cert_pwd',
158     location      => '\\\\OFFUSCATEDPATH\\TerminalServices\\certificates',
159     thumbprint    => $cert_thumbprint,
160 }
```



---

```

158
159 # check certificate permissions and reset to self-signed certificate
160 exec {'check cert permissions':
161     command => "$Thumbprint = (Get-ChildItem cert:\\LocalMachine\\My | Where-Object
    ↳ {$_Subject -match '\\$env:COMPUTERNAME.cern.ch'}).Thumbprint;
162     Get-WmiObject -Class Win32_TSGeneralSetting -Namespace
    ↳ root\\cimv2\\TerminalServices | Set-WmiInstance -Arguments
    ↳ @{'SSLCertificateSHA1Hash' = '$Thumbprint'}";
163     onlyif => "$keyName = (((Get-ChildItem Cert:\\LocalMachine\\My | Where-Object
    ↳ {$_Thumbprint -like
    ↳ '$cert_thumbprint'}).PrivateKey).CspKeyContainerInfo).UniqueKeyContainerName;
164     $keyPath = 'C:\\ProgramData\\Microsoft\\Crypto\\RSA\\MachineKeys';
165     $fullPath = $keyPath + '\\\\' + $keyName;
166     $aclNetSvc = (Get-Acl -Path $fullPath).Access | Where-Object
    ↳ {$_IdentityReference -match 'NETWORK SERVICE'}
167     If ($aclNetSvc -ne $null -and $aclNetSvc.AccessControlType -eq 'Allow' -and
    ↳ $aclNetSvc.FileSystemRights -match 'Read')
168     { exit 1 }
169     else
170     { exit 0 }",
171     provider => powershell,
172     require => Cernsslcertificate['INSTALL-CERTIFICATE']
173 }
174
175 # Install SSL Certificate in RD Session Host for RDP-TCP Connections
176 wmi { 'Install SSL Certificate for RDP':
177     wmi_namespace => 'root/cimv2/terminalservices',
178     wmi_class => 'Win32_TSGeneralSetting',
179     wmi_property => 'SSLCertificateSHA1Hash',
180     wmi_value => $cert_thumbprint, # Correct Certificate thumbprint,
181     require => Exec['check cert permissions']
182 }
183
184 }
185 }

```

---



## PUPPET MANIFEST TO CONFIGURE REMOTE DESKTOP LICENSE SERVER

```
1 class hg_windows_infrastructure::rds::win2012::lic {
2   if $::operatingsystem == 'windows' {
3
4     exec { 'Disable Firewall':
5       command => 'netsh advfirewall set allprofiles state off',
6       unless   => '$FirewallState = netsh advfirewall show domainprofile state |
7         ↪ Select-String "State"; if($FirewallState -like "*OFF") { exit 0 } else { exit 1
8         ↪ }',
9       provider => powershell
10    }
11
12    # Install Remote Desktop Licensing
13    windowsfeature { 'RDS-Licensing':
14      ensure => present,
15      restart => true
16    }
17
18    # Install Remote Desktop Connection Broker
19    windowsfeature { 'RDS-Connection-Broker':
20      ensure => present,
21      restart => true
22    }
23
24    # Install Remote Desktop Licensing Tools
25    windowsfeature { 'RDS-Licensing-UI':
26      ensure => present,
27      restart => true
28    }
29  }
```

## ANNEX XIX. PUPPET MANIFEST TO CONFIGURE REMOTE DESKTOP LICENSE SERVER

---

```
27
28   # Remote Desktop Services enable license server security group
29   dsc_administrativetemplatesetting { 'Enable license server security group':
30     dsc_keyvaluename => 'SOFTWARE\Policies\Microsoft\Windows NT\Terminal
      ↳ Services\SecureLicensing',
31     dsc_policytype  => 'Machine',
32     dsc_data        => '1', # Enable
33     dsc_ensure      => 'Present',
34     dsc_type        => 4,
35   }
36
37   # Only members of this group can obtain RDS licenses and connect to the broker
38   group { 'RDS Endpoint Servers group membership':
39     name      => 'RDS Endpoint Servers',
40     ensure    => present,
41     members   => ['cern\NICE Terminal Services Application Servers'],
42     auth_membership => false,
43   }
44 }
45 }
46 }
```

---



## SYNC-FILES POWERSHELL SCRIPT TO SYNCHRONISE A FOLDER STRUCTURE

---

```

1 Param(
2 [Parameter(Mandatory=$true)][string]$SourceDirPath,
3 [Parameter(Mandatory=$true)][string]$TargetDirPath
4 )
5 # Exit codes for Puppet: Exit 0 - Success (no errors); Exit 1 - Failed (errors)
6
7 Function Compute-Hash {
8
9 Param($Path, $Algorithm )
10
11     if ($PSVersionTable.PSVersion.Major -ge 4) {
12         return $(Get-FileHash -Path $Path -Algorithm $Algorithm).Hash
13     } else {
14         $hash = [Security.Cryptography.HashAlgorithm]::Create( $Algorithm )
15         $stream = ([IO.StreamReader]"$Path").BaseStream
16         $output = -join ($hash.ComputeHash($stream) | ForEach { "{0:x2}" -f $_ })
17         $stream.Close()
18         $object = New-Object -TypeName psobject -Property @{
19             Algorithm = $Algorithm
20             Hash = $output.ToUpper()
21             Path = $Path
22         }
23         return $object
24     }
25 }
26 }
27
28 #source dir must exist

```

## ANNEX XX. SYNC-FILES POWERSHELL SCRIPT TO SYNCHRONISE A FOLDER STRUCTURE

---

```
29  if(!(Test-Path $SourceDirPath)){
30      exit 1
31  }
32  # ! $_.PSIsContainer ignores directories
33  $SourceDirFiles = Get-ChildItem -Recurse -Path $SourceDirPath | where { !
    ↳ $_.PSIsContainer }
34
35  # target dir exists
36  if(Test-Path $TargetDirPath){
37      $TargetDirFiles = Get-ChildItem -Recurse -Path $TargetDirPath | where { !
    ↳ $_.PSIsContainer }
38      if($TargetDirFiles){
39
40          $TargetDirFileHashes = $TargetDirFiles | foreach {Compute-Hash -Path $_.FullName
    ↳ -Algorithm MD5}
41
42          # Compares file names
43          $FilesDiff = Compare-Object -ReferenceObject $SourceDirFiles -DifferenceObject
    ↳ $TargetDirFiles
44
45          # Compares file contents
46          $FileHashesDiff = Compare-Object -ReferenceObject $SourceDirFileHashes
    ↳ -DifferenceObject $TargetDirFileHashes
47
48          if(($FilesDiff.SideIndicator.Length -eq 0) -and
    ↳ ($FileHashesDiff.SideIndicator.Length -eq 0)){
49              #echo 'empty - no diffs! Do nothing! exiting...' #TODO DEBUG
50              exit 0
51          } else {
52              #echo 'different trees. Deleting dir ...' #TODO DEBUG
53              Remove-Item $TargetDirPath\* -Recurse -Force
54          }
55      }
56  }
57  Copy-Item $SourceDirPath\* -Destination $TargetDirPath -Recurse
58  exit 0
```

---



## CERN OPERATIONS SMA INTEGRATION MODULE

---

```

1 <#
2   .SYNOPSIS
3   Common functions for CERN Operations
4   Last Update: 26/10/2017
5   Authors: Sebastian Buckowiec, Ricardo Gaspar
6
7   Disclaimer: Not the original version! Using non-real CERN users, URIs and server
   ↪ addresses for security purposes.
8 #>
9
10 # Global Variables
11 $CMFProductionSOAPURI = "https://someweburiatcern.asmx?WSDL" #production
12
13 function Test-Health {
14     param(
15         [Parameter(Mandatory = $True)]
16         [ValidateNotNullOrEmpty()]
17         [String]
18         $ComputerName,
19
20         [Parameter(Mandatory = $False)]
21         [ValidateNotNullOrEmpty()]
22         [int]
23         $Port,
24
25         [Parameter(Mandatory = $False)]
26         [ValidateNotNullOrEmpty()]
27         [int]

```

```
28     $Fall = 3
29 )
30
31 Clear-DnsClientCache
32
33 if (!$Port) {
34     $cmd = "Test-NetConnection -ComputerName $ComputerName -WarningAction
35         ↪ SilentlyContinue -InformationLevel Quiet"
36 }
37 else {
38     $cmd = "Test-NetConnection -ComputerName $ComputerName -Port $Port -WarningAction
39         ↪ SilentlyContinue -InformationLevel Quiet"
40 }
41
42 $TestResult = Invoke-Expression $cmd
43 if (!$TestResult) {
44     for ($i = 1; $i -lt $Fall; $i++) {
45         Start-Sleep 3
46         Clear-DnsClientCache
47         $TestResult = Invoke-Expression $cmd
48         if ($TestResult) {
49             return $True
50         }
51     }
52 }
53 else {
54     return $True
55 }
56
57 return $False
58 }
59
60 function Send-Notification {
61     param(
62         [Parameter(Mandatory = $True)]
63         [ValidateNotNullOrEmpty()]
64         [String]
65         $To,
66
67         [Parameter(Mandatory = $True)]
68         [ValidateNotNullOrEmpty()]
69         [String]
70         $Subject,
71
72         [Parameter(Mandatory = $True)]
73         [ValidateNotNullOrEmpty()]
74         [String]
75         $Body
76     )
77 }
```



---

```

76     $notifRecipients = $To -split ", " | % { $_.trim() }
77
78     Send-MailMessage -To $notifRecipients -From "someadminuser@cern.ch" -Subject $Subject
    ↪ -Body $Body -SmtpServer somecernmailserver.cern.ch
79
80 }
81
82
83 <#
84 .SYNOPSIS
85 DeleteVolume function deletes a volume based on its ID.
86
87 .DESCRIPTION
88 DeleteVolume function deletes a volume based on its ID.
89 It conctacts the administration server (aiadm) in other to invoke the 'openstack
    ↪ volume delete' command that deletes the volume on OpenStack.
90
91 This operation should be done on volumes that are not attached to any VM.
92
93 It returns the message retruned by openstack command.
94
95 .PARAMETER VolumeID
96 The volume identifier on OpenStack.
97
98 .PARAMETER Connection
99 The connection object to be used. It contains the user and password to be used and
    ↪ the server name to connect to.
100
101 .PARAMETER OpenStackProject
102 The OpenStack project under which the volume was created.
103
104 .EXAMPLE
105 $deleteVolResult = DeleteVolume -VolumeID "7c123423-5ca0-495f-b84a-69db5a76541c"
    ↪ -Connection $Conn -OpenStackProject "my openstack project"
106
107 .NOTES
108 This operation should be done on volumes that are not attached to any VM.
109 #>
110 function DeleteVolume {
111     [CmdletBinding(DefaultParameterSetName = 'SpecifyConnectionFields')]
112     param(
113         [Parameter(Mandatory = $True)]
114         [ValidateNotNullOrEmpty()]
115         [string] $VolumeID,
116
117         [Parameter(ParameterSetName = 'UseConnectionObject', Mandatory = $True)]
118         [ValidateNotNullOrEmpty()]
119         [Hashtable]
120         $Connection,
121

```

```
122     [Parameter(Mandatory = $True)]
123     [ValidateNotNullOrEmpty()]
124     [string] $OpenStackProject
125 )
126
127 $deleteVolumeCmd = [Scriptblock]::Create("eval `$(ai-rc $OpenStackProject); openstack
↪ volume delete $VolumeID")
128 $deleteVolumeCmdResult = Invoke-SSHCommand -Connection $Connection -ScriptBlock
↪ $deleteVolumeCmd
129 return $deleteVolumeCmdResult.Result
130 }
131
132
133 <#
134 .SYNOPSIS
135     DeleteVM function deletes a VM based on its name.
136
137 .DESCRIPTION
138     DeleteVM function deletes a VM based on its name on OpenStack.
139     It conctacts the administration server (aiadm) in other to invoke the ai-kill command
↪ line tool that kills the VM on OpenStack and cleans its records on Foreman.
140
141     It returns the message retruned by ai-kill command.
142 .PARAMETER VMName
143     The name of the VM.
144
145 .PARAMETER Connection
146     The connection object to be used. It contains the user and password to be used and
↪ the server name to connect to.
147
148 .PARAMETER OpenStackProject
149     The OpenStack project under which the volume was created.
150
151 .EXAMPLE
152     $deleteVMResult = DeleteVm -VMName "mycomputername" -Connection $Conn
↪ -OpenStackProject "my openstack project"
153
154 #>
155
156 function DeleteVM {
157     [CmdletBinding(DefaultParameterSetName = 'SpecifyConnectionFields')]
158     param(
159         [Parameter(Mandatory = $True)]
160         [ValidateNotNullOrEmpty()]
161         [string] $VMName,
162
163         [Parameter(ParameterSetName = 'UseConnectionObject', Mandatory = $True)]
164         [ValidateNotNullOrEmpty()]
165         [Hashtable]
166         $Connection,
```

---

```

167
168     [Parameter(Mandatory = $True)]
169     [ValidateNotNullOrEmpty()]
170     [string] $OpenStackProject
171 )
172
173 $deleteVMCmd = [Scriptblock]::Create("eval `$(ai-rc $OpenStackProject); ai-kill-vm
↪ $VMName")
174 $deleteVMCmdResult = Invoke-SSHCommand -Connection $Connection -ScriptBlock
↪ $deleteVMCmd
175 return $deleteVMCmdResult.Result
176 }
177
178 <#
179 .SYNOPSIS
180 WaitForVM function waits for the VM to be reachable through WinRM.
181
182 .DESCRIPTION
183 WaitForVM function waits for the VM to be reachable through WinRM.
184 It tries to reach the VM multiple times, using a given sleep interval, until the VM
↪ responds.
185 First it waits for the VM to be in an 'ACTIVE' state on OpenStack, if it reaches the
↪ 'ERROR' state then the function throws an exception indicating that the VM was not
↪ created, exception message: "[WaitForVM function] an error occurred when creating the
↪ VM on OpenStack."
186
187 After the VM reaches the 'ACTIVE' state, then the function waits for the VM to be
↪ reachable through WinRM (Windows Remote Management Protocol).
188
189 At the end it returns the total time slept until the machine is running and
↪ reachable by WinRM.
190
191 .PARAMETER SleepInterval
192 The time in seconds between each connection attempt.
193
194 .PARAMETER VMName
195 The name of the VM.
196
197 .PARAMETER Connection
198 The connection object to be used. It contains the user and password to be used and
↪ the server name to connect to.
199
200 .PARAMETER OpenStackProject
201 The OpenStack project under which the VM was created.
202
203 .EXAMPLE
204 Sleep for 300 secs = 5 min, until VM is ready to receive commands
205 WaitForVM -SleepInterval 300 -VMName "mycomputername" -Connection $Connection
↪ -OpenStackProject "my openstack project"
206

```

```
207 .EXAMPLE
208 try {
209     $totalWaitTime = WaitForVM -SleepInterval 300 -VMName "mycomputername"
    ↪ -Connection $Connection -OpenStackProject "my openstack project" # Sleep for 300 secs
    ↪ = 5 min, until VM is ready to receive commands
210 }
211 catch {
212     Write-Output "An error occured when creating the VM on OpenStack."
213 }
214 #>
215
216 function WaitForVM {
217     [CmdletBinding(DefaultParameterSetName = 'SpecifyConnectionFields')]
218     param(
219         [Parameter(Mandatory = $True)]
220         [ValidateNotNullOrEmpty()]
221         [int] $SleepInterval,
222
223         [Parameter(Mandatory = $True)]
224         [ValidateNotNullOrEmpty()]
225         [string] $VMName,
226
227         [Parameter(ParameterSetName = 'UseConnectionObject', Mandatory = $True)]
228         [ValidateNotNullOrEmpty()]
229         [Hashtable]
230         $Connection,
231
232         [Parameter(Mandatory = $True)]
233         [ValidateNotNullOrEmpty()]
234         [string] $OpenStackProject
235     )
236
237     $totalTimeSlept = 0
238     $vmStatusCmd = [Scriptblock]::Create("eval `$(ai-rc $OpenStackProject); openstack
    ↪ server show $VMName -c status -f value")
239
240     Write-Verbose "[WaitForVM function] Waiting for the VM to be ACTIVE."
241     # wait for VM to be running (OpenStack status = ACTIVE)
242     Do {
243         $vmStatus = Invoke-SSHCommand -Connection $Connection -ScriptBlock $vmStatusCmd
244         $vmStatusResult = ($vmStatus.Result).Trim().ToUpper()
245
246         switch ($vmStatusResult) {
247             "ERROR" {
248                 Write-Verbose "[WaitForVM function] an error occured when creating the VM
                ↪ on OpenStack. vmStatusResult = $vmStatusResult."
249                 throw "[WaitForVM function] an error occured when creating the VM on
                ↪ OpenStack."
250             }
251             "ACTIVE" {
```

---

```

252         Write-Verbose "[WaitForVM function] VM is now ACTIVE, total time slept so
           ↳ far = $totalTimeSlept."
253         break
254     }
255     Default {
256         Write-Verbose "[WaitForVM function] VM is being created (BUILD). Sleeping
           ↳ for $SleepInterval seconds."
257         Start-Sleep -Seconds $SleepInterval
258         $totalTimeSlept += $SleepInterval
259     }
260 }
261 } Until ($vmStatusResult -eq "ACTIVE")
262
263 Write-Verbose "[WaitForVM function] Waiting for the VM to be reachable by WinRM."
264 # wait for VM to be reachable by WinRM
265 Do {
266     $winRMconnection = Test-Wsman $VMName -ErrorAction SilentlyContinue
267     if ($winRMconnection) {
268         Write-Verbose "[WaitForVM function] WinRM OK! `r`n total time slept:
           ↳ $totalTimeSlept seconds."
269     }
270     else {
271         Write-Verbose "[WaitForVM function] WinRM not yet available. Sleeping for
           ↳ $SleepInterval seconds."
272         Start-Sleep -Seconds $SleepInterval
273         $totalTimeSlept += $SleepInterval
274     }
275 } Until ($winRMconnection)
276
277 return $totalTimeSlept
278 }
279
280
281 <#
282 .SYNOPSIS
283 WaitForCMF function Waits for the VM's CMF agent to finish the deployment of pending
   ↳ packages.
284
285 .DESCRIPTION
286 WaitForCMF function Waits for the VM's CMF agent to finish the deployment of pending
   ↳ packages.
287 It tries multiple times, using a given sleep interval, until the CMF indicates that
   ↳ it completed all the installations.
288
289 .PARAMETER SleepInterval
290 The time in seconds between each connection attempt.
291
292 .PARAMETER VMName
293 The name of the VM.
294

```

```
295 .EXAMPLE
296 $totalWaitTimeCMF = WaitForCMF -SleepInterval 180 -VMName "mycomputername"
297
298 .EXAMPLE
299 WaitForCMF -SleepInterval 180 -VMName "mycomputername"
300 #>
301 function WaitForCMF {
302     param(
303         [Parameter(Mandatory = $True)]
304         [ValidateNotNullOrEmpty()]
305         [int] $SleepInterval,
306
307         [Parameter(Mandatory = $True)]
308         [ValidateNotNullOrEmpty()]
309         [string] $VMName
310     )
311
312     Write-Verbose "[WaitForCMF function]"
313     $totalTimeSlept = 0
314
315     $URI = $CMFProductionSOAPURI
316
317     $CMFWebSvc = New-WebServiceProxy -Uri $URI -namespace WebServiceProxy -Class
318     ↪ NSCMgtSoap
319
320     Do {
321         Write-Verbose "[WaitForCMF function] Getting status of the computer $VMName"
322         $CMFStatusCompleted = $CMFWebSvc.CMFCompleted($VMName)
323         if (!$CMFStatusCompleted) {
324             Write-Verbose "[WaitForCMF function] CMF is NOT finished yet! Sleeping for
325             ↪ $SleepInterval seconds."
326             Start-Sleep -Seconds $SleepInterval
327             $totalTimeSlept += $SleepInterval
328         }
329         else {
330             Write-Verbose "[WaitForCMF function] CMF finished!"
331         }
332     } Until ($CMFStatusCompleted)
333
334     return $totalTimeSlept
335 }
336
337 <#
338 .SYNOPSIS
339 Copies a file or an directory available locally on the SMA worker or in a network path to
340 ↪ a target computer.
341
342 .DESCRIPTION
343 Copies a file or an entire directory available locally on the SMA worker or in a network
344 ↪ path to a target computer.
345
346 .PARAMETER ComputerName
```

---

```

341 Name of the computer to copy the file.
342 .PARAMETER SourcePath
343 Path to the file or directory to be copied.
344 Path to files must include the file extension (see examples).
345 Path to directory should include the last slash (\) (see examples).
346 .PARAMETER TargetDirPath
347 Path to the directory where the file or directory must be copied on the target computer.
348 In case of copying a directory, it will copy the whole directory (including sub-files and
    ↪ sub-directories) specified in the SourcePath into the directory specified in
    ↪ TargetDirPath.
349 Default value : "C:\Windows\Temp\SMA_files"
350 .PARAMETER Credential
351 PSCredential object containing the credentials with authotized access to invoke
    ↪ operations on target computer.
352 .EXAMPLE
353 # Copy a single file
354 Copy-RemoteFiles -ComputerName "myservername" -SourcePath "\\fileserver\file.ps1"
    ↪ -Credential $Cred
355 .EXAMPLE
356 # Copy a single file specifying the target directory
357 Copy-RemoteFiles -ComputerName "myservername" -SourcePath "\\fileserver\file.ps1"
    ↪ -TargetDirPath "C:\mydir" -Credential $Cred
358 .EXAMPLE
359 # Copy a directory
360 Copy-RemoteFiles -ComputerName "myservername" -SourcePath "\\fileserver\directory\"
    ↪ -Credential $Cred
361 .EXAMPLE
362 # Copy a directory specifying the target directory
363 Copy-RemoteFiles -ComputerName "myservername" -SourcePath "\\fileserver\directory\"
    ↪ -TargetDirPath "C:\mydir" -Credential $Cred
364 #>
365 function Copy-RemoteFiles {
366     param(
367         [Parameter(Mandatory = $True)]
368         [ValidateNotNullOrEmpty()]
369         [string] $ComputerName,
370
371         [Parameter(Mandatory = $True)]
372         [ValidateNotNullOrEmpty()]
373         [string] $SourcePath,
374
375         [string] $TargetDirPath = "C:\Windows\Temp\SMA_files",
376
377         [Parameter(Mandatory = $True)]
378         [ValidateNotNullOrEmpty()]
379         [PSCredential] $Credential
380     )
381
382     $session = New-PSSession -ComputerName $ComputerName -Credential $Credential

```

```
384
385     $sourceItem = Get-Item $SourcePath
386     $targetFilePath = Join-Path $TargetDirPath $sourceItem.Name
387
388     Write-Verbose "targetFilePath=$targetFilePath"
389
390     # create directory if it does not exist, if it exists there are no side effects
391     $createScriptsDirCMD = [Scriptblock]::Create("New-Item -ItemType directory -Path
↪ $TargetDirPath -Force")
392     $createScriptsDirCMDResult = Invoke-Command -Session $session -ScriptBlock
↪ $createScriptsDirCMD
393
394     Write-Verbose "$createScriptsDirCMDResult"
395
396     try {
397         # case insensitive switch
398         switch ($sourceItem.GetType().Name) {
399             "FileInfo" {
400                 $copyResult = Copy-Item -Path $SourcePath -Destination $targetFilePath
↪ -ToSession $session
401             }
402             "DirectoryInfo" {
403                 $copyResult = Copy-Item -Path $SourcePath -Destination $targetFilePath
↪ -ToSession $session -Recurse
404             }
405         }
406         Write-Verbose "$copyResult"
407         Write-Verbose "file $SourcePath copied to $TargetDirPath"
408     }
409     catch {
410         Write-Verbose "An error occurred when copying $SourcePath to $TargetDirPath"
411     }
412
413     Remove-PSSession $session
414 }
415
416 <#
417 .SYNOPSIS
418 Copies a executable file available locally on the SMA worker or in a network path to a
↪ target computer and executes it on the target computer.
419
420 .DESCRIPTION
421 Copies a executable file available locally on the SMA worker or in a network path to a
↪ target computer and executes it on the target computer.
422
423 .PARAMETER ComputerName
424 Name of the computer to copy the file.
425
426 .PARAMETER SourcePath
427 Path to the executable file to be copied and executed.
```



---

```

428
429 .PARAMETER TargetDirPath
430 Path to the directory where the executable file must be copied on the target computer.
431 Default is : "C:\Windows\Temp\SMA_files"
432
433 .PARAMETER TargetScriptArguments
434 A string with the arguments to pass to the script.
435
436 .PARAMETER DeleteFilesAfterwards
437 When specified, deletes the executable file after it has been executed.
438
439 .PARAMETER Credential
440 PSCredential object containing the credentials with authotized access to invoke
441 ↪ operations on target computer.
442
443 .EXAMPLE
444 Copy-ExecuteScript -ComputerName "myservername" -SourcePath "\\filesrv\file.ps1"
445 ↪ -TargetDirPath -Credential $Cred
446
447 .EXAMPLE
448 Copy-ExecuteScript -ComputerName "myservername" -SourcePath "\\filesrv\file.ps1"
449 ↪ -TargetDirPath "C:\Windows\Temp\SMA_files" -TargetScriptArguments "-AnArgument x"
450 ↪ -Credential $Cred -DeleteFilesAfterwards
451
452 #>
453 function Copy-ExecuteScript {
454     param(
455         [Parameter(Mandatory = $True)]
456         [ValidateNotNullOrEmpty()]
457         [string] $ComputerName,
458
459         [Parameter(Mandatory = $True)]
460         [ValidateNotNullOrEmpty()]
461         [string] $SourcePath,
462
463         [string] $TargetDirPath = "C:\Windows\Temp\SMA_files",
464
465         [string] $TargetScriptArguments,
466
467         [switch] $DeleteFilesAfterwards,
468
469         [Parameter(Mandatory = $True)]
470         [ValidateNotNullOrEmpty()]
471         [PSCredential] $Credential
472     )
473
474     $session = New-PSSession -ComputerName $ComputerName -Credential $Credential
475
476     Copy-RemoteFiles -ComputerName $ComputerName -SourcePath $SourcePath -TargetDirPath
477     ↪ $TargetDirPath -Credential $Credential

```

```
473
474     $sourceScriptPath = Get-Item $SourcePath
475     $targetScriptPath = Join-Path $TargetDirPath $sourceScriptPath.Name
476
477     if ($TargetScriptArguments) {
478         $targetScriptCommand = "$targetScriptPath $TargetScriptArguments"
479     }
480     else {
481         $targetScriptCommand = $targetScriptPath
482     }
483
484     if ($DeleteFilesAfterwards) {
485         $executeScriptCMD = [Scriptblock]::Create("$targetScriptCommand; Remove-Item
486             ↳ $TargetDirPath -Recurse -Force")
487     }
488     else {
489         $executeScriptCMD = [Scriptblock]::Create("$targetScriptCommand")
490     }
491
492     Write-Verbose "Running Command: `r`n $($executeScriptCMD.ToString())"
493
494     Invoke-Command -Session $session -ScriptBlock $executeScriptCMD
495
496     Remove-PSSession $session
497 }
498 <#
499 .SYNOPSIS
500     Regenerates puppet certificate on the target computer.
501
502 .DESCRIPTION
503     Regenerates puppet certificate on the target computer.
504
505 .PARAMETER ComputerName
506     Name of the computer to regenerate the puppet certificate.
507
508 .PARAMETER Credential
509     PSCredential object containing the credentials with authotized access to invoke
510     ↳ operations on target computer.
511
512 .EXAMPLE
513     FixPuppetCert -ComputerName "myservername" -Credential $Cred
514
515 #>
516 function FixPuppetCert {
517     param(
518         [Parameter(Mandatory = $True)]
519         [ValidateNotNullOrEmpty()]
520         [string] $ComputerName,
```

---

```

521     [Parameter(Mandatory = $True)]
522     [ValidateNotNullOrEmpty()]
523     [PSCredential] $Credential
524
525 )
526
527 $OpenSSLDirPath = "\\filesrv\openssl\"
528 $GenCertScriptPath = "\\filesrv\certfixscript.ps1"
529 $TargetDirPath = "C:\Windows\Temp\SMA_files\"
530
531 $session = New-PSSession -ComputerName $ComputerName -Credential $Credential
532
533 # Copy Openssl files required by the certificate regenerator script
534 Copy-RemoteFiles -ComputerName $ComputerName -SourcePath $OpenSSLDirPath
535 ↪ -TargetDirPath $TargetDirPath -Credential $Credential
536
537 # Copy and execute certificate regenerator script
538 Copy-ExecuteScript -ComputerName $ComputerName -SourcePath $GenCertScriptPath
539 ↪ -TargetDirPath $TargetDirPath -TargetScriptArguments "-OpenSSExecutablePath
540 ↪ $TargetDirPath\openssl\openssl.exe" -Credential $Credential
541
542 Remove-PSSession $session
543 }
544 <#
545 .SYNOPSIS
546 Add a computer to a given CMF Name Set of Computers (NSC).
547 .DESCRIPTION
548 Add a computer to a given CMF Name Set of Computers (NSC).
549
550 .PARAMETER ComputerName
551 Name of the computer to add to the NSC.
552
553 .PARAMETER NSCid
554 The number ID of the NSC.
555
556 .PARAMETER Credential
557 PSCredential object containing the credentials with authorized access to invoke
558 ↪ operations on target computer.
559
560 .EXAMPLE
561 CMFAddComputerToNSC -ComputerName "myservername" -NSCid "1059" -Credential $Cred
562
563 .NOTES
564 The add to NSC operation is and MUST be invoked from the target computer in order to be
565 ↪ successful.
566 #>
567 function CMFAddComputerToNSC {

```

```
566 Param(
567     [Parameter(Mandatory = $true)]
568     [string] $ComputerName,
569
570     [Parameter(Mandatory = $true)]
571     [string] $NSCid,
572
573     [Parameter(Mandatory = $True)]
574     [ValidateNotNullOrEmpty()]
575     [PSCredential] $Credential
576 )
577 $URI = $CMFProductionSOAPURI
578
579
580 $session = New-PSSession -ComputerName $ComputerName -Credential $Credential
581
582 # IMPORTANT: In order to work you run this script from the Computer you want to add
583 ↪ to NSC.
584 Write-Verbose "Adding the computer $ComputerName to NSC $NSCid"
585
586 Invoke-Command -Session $session -ScriptBlock {$CMFWebSvc = New-WebServiceProxy -Uri
587     ↪ $Using:URI -namespace WebServiceProxy -Class NSCMgtSoap
588     $result = $CMFWebSvc.AddComputerToNSC($Using:ComputerName, $Using:NSCid)
589     if ($result) {
590         Write-Output "The following error occurred:"
591         $result
592     }
593     else {
594         Write-Output "Computer $Using:ComputerName successfully added to NSC
595         ↪ $Using:NSCid"
596     }
597 }
598
599 Remove-PSSession $session
600 }
601
602 <#
603 .SYNOPSIS
604 Trigger the CMF agent of a computer to install the scheduled software packages.
605
606 .DESCRIPTION
607 Trigger the CMF agent of a computer to install the scheduled software packages.
608
609 .PARAMETER ComputerName
610 Name of the computer to trigger the execution of pending actions (install CMF packages).
611
612 .EXAMPLE
613 CMFExecPendingActions -ComputerName "myservername"
614 #>
```

---

```

613 function CMFExecPendingActions {
614     Param(
615         [Parameter(Mandatory = $true)]
616         [string] $ComputerName
617     )
618
619     $URI = $CMFProductionSOAPURI
620
621     $CMFWebSvc = New-WebServiceProxy -Uri $URI -namespace WebServiceProxy -Class
        ↳ NSCMgtSoap
622
623     Write-Verbose "Asking CMF agent of computer $ComputerName to execute the pending
        ↳ actions."
624     $result = $CMFWebSvc.ExecutependingActions($ComputerName)
625
626     if ($result) {
627         Write-Output "The following error occurred:`r`n $result"
628     }
629     else {
630         Write-Output "Computer $ComputerName is now executing the pending actions."
631     }
632
633 }
634
635 <#
636 .SYNOPSIS
637 Get a list of the software packages scheduled to be installed on a computer.
638
639 .DESCRIPTION
640 Get a list of the software packages scheduled to be installed on a computer in a string
        ↳ object.
641 If the computer does not exist returns $null.
642
643 .PARAMETER ComputerName
644 Name of the computer to get the list of missing software packages.
645
646 .PARAMETER ReturnValuesOnly
647 Flag parameter to supress any output messages and get only the result values.
648 When specified, no messages are displayed in the console and only returns the string
        ↳ object containing the missing software packages. In this case, if there are no
        ↳ packages to install a $null value is returned.
649
650 .EXAMPLE
651 Example with output messages.
652 CMFGetPendingActions -ComputerName "myservername"
653
654 .EXAMPLE
655 Example returning only the packages in a string object.
656 CMFGetPendingActions -ComputerName "myservername" -ReturnValuesOnly
657

```

```
658 .NOTES
659 The ReturnValuesOnly flag is intended to be used when some other program needs to do
    ↳ something with the list of packages (string object) because, in PowerShell, the
    ↳ messages written to the console are also returned as values by the functions. With
    ↳ this flag parameter only the value is returned so that developers don't have to parse
    ↳ or filter the returned values.
660 #>
661 function CMFGetPendingActions {
662     Param(
663         [Parameter(Mandatory = $true)]
664         [string] $ComputerName,
665
666         [switch] $ReturnValuesOnly
667     )
668
669     $URI = $CMFProductionSOAPURI
670
671     $CMFWebSvc = New-WebServiceProxy -Uri $URI -namespace WebServiceProxy -Class
    ↳ NSCMgtSoap
672
673     $pendingActions = $CMFWebSvc.GetPendingActions($ComputerName)
674
675     # in case of just wanting the values ($null if no value)
676     if ($ReturnValuesOnly) {
677         return $pendingActions
678     }
679
680     if (!$pendingActions) {
681         Write-Output "There are no pending actions!"
682     }
683     else {
684         # Display each package in each line
685         $pendingActionsString = ""
686         foreach ($item in $pendingActions) {
687             $pendingActionsString += "- $item`r`n"
688         }
689         Write-Output "There are pending actions:"
690         $pendingActionsString
691     }
692 }
693
694 <#
695 .SYNOPSIS
696 Get the CMF Agent status of a computer.
697
698 .DESCRIPTION
699 Get the CMF Agent status of a computer.
700 Returns a boolean value: $true if all the software packages are installed and $false if
    ↳ the CMF agent is still installing packages.
701 If the computer does not exist returns $null.
```

---

```

702
703 .PARAMETER ComputerName
704 Name of the computer to check CMF Agent status.
705
706 .PARAMETER ReturnValuesOnly
707 Flag parameter to suppress any output messages and get only the result value.
708 When specified, no messages are displayed in the console and only returns the boolean
    ↳ value that indicates if the CMF Agent finished installing the software packages or
    ↳ not.
709
710 .EXAMPLE
711 Example with output messages.
712 CMFGetStatus -ComputerName "myservername"
713
714 .EXAMPLE
715 Example returning only the status in a boolean result.
716 CMFGetStatus -ComputerName "myservername" -ReturnValuesOnly
717
718 .NOTES
719 The ReturnValuesOnly flag is intended to be used when some other program needs to do
    ↳ something with the result value (boolean) because, in PowerShell, the messages
    ↳ written to the console are also returned as values by the functions. With this flag
    ↳ parameter only the value is returned so that developers don't have to parse or filter
    ↳ the returned values.
720 #>
721 function CMFGetStatus {
722     Param(
723         [Parameter(Mandatory = $true)]
724         [string] $ComputerName,
725
726         [switch] $ReturnValuesOnly
727     )
728
729     $URI = $CMFProductionSOAPURI
730
731     $CMFWebSvc = New-WebServiceProxy -Uri $URI -namespace WebServiceProxy -Class
    ↳ NSCMgtSoap
732
733     $CMFComputerStatus = $CMFWebSvc.CMFCompleted($ComputerName)
734
735     if ($ReturnValuesOnly) {
736         return $CMFComputerStatus
737     }
738
739     if (!$CMFComputerStatus) {
740         Write-Output "CMF is NOT finished yet!"
741     }
742     else {
743         Write-Output "CMF finished!"
744     }

```

745 }

746

747 `Export-ModuleMember *`

---



## create-vm-with-volume RUNBOOK

---

```

1 <#
2 .SYNOPSIS
3 Workflow to create CERN Windows VM with a second volume attached.
4
5 .DESCRIPTION
6 Workflow designed to create new Windows VMs in CERN infrastructure with a second volume
7 ↪ attached.
8 It uses ai-bs-vm commandline tool to create the VM in OpenStack, register in Foreman
9 ↪ hostgroup and add it to NSCs on CMF.
10
11 .NOTES
12 Author: Ricardo Gaspar (ricardo.gaspar@cern.ch)
13 Last update: 25/10/2017
14
15 .PARAMETER OpenStackProject
16 'OpenStackProject' is the OpenStack project where the VM will be created.
17
18 .PARAMETER Hostgroup
19 'Hostgroup' is the Foreman hostgroup where the VM will be placed.
20
21 .PARAMETER Environment
22 'Environment' is the Foreman environment where the VM will be placed.
23 Common values can be: 'qa','production'.
24 Default value: 'production'
25
26 .PARAMETER NovaImage
27 'NovaImage' is the OS image name available on OpenStack Nova to be installed on the VM.
28 Check the available images on OpenStack or run 'openstack image list' in AIADM.
29 Default value: 'Windows 2016 Standard'

```

29 **.PARAMETER** NovaFlavor  
30 'NovaFlavor' is the OpenStack Nova VM flavour. Windows machines require a minimum of 40GB  
↪ of disk space, so 'm2.large' flavor is the minimum flavor.  
31 Check the available flavors on OpenStack or run 'openstack flavor list' in AIADM.  
32 Default value: 'm2.xlarge'  
33  
34 **.PARAMETER** LANDBResponsible  
35 'LANDBResponsible' is the LAN DB responsible user/group (main responsible) for the VM.  
↪ The user/group must exist in e-groups and Active Directory.  
36 Default value: 'SUPPORT-WINDOWS-SERVERS'  
37  
38 **.PARAMETER** LANDBMainUser  
39 'LANDBMainUser' is the LAN DB main user user/group for the VM. The user/group must exist  
↪ in e-groups and Active Directory.  
40 Default value: 'bukowiec'  
41  
42 **.PARAMETER** AvailabilityZone  
43 'AvailabilityZone' is the datacenter's availability zone where the VM will be placed.  
44 Check the existing availability zones on OpenStack or run 'openstack availability zone  
↪ list' in AIADM.  
45 Default value: 'cern-geneva-a'  
46  
47 **.PARAMETER** CMFMemberships  
48 'CMFMemberships' is a list of the CMF NSCs to which VM must be member. This parameter is  
↪ a list of strings separated by commas.  
49 Example: '1059,1304'  
50 Default value: '1059'  
51  
52 **.PARAMETER** VolumeSize  
53 'VolumeSize' is the size of the volume to be created and attached to the VM.  
54 This parameter follows the same structure as the --nova-attach-new-volume argument of  
↪ 'ai-bs-vm' tool.  
55 For more info check the manual page: 'man ai-bs-vm' in AIADM.  
56 Values should be like "xGB" or "xTB" (1TB = 1000GB), where x is an integer > 0.  
57 Example: '1GB'  
58 Default value: '100GB'  
59  
60 **.PARAMETER** VolumeType  
61 'VolumeType' is the type of the volume to be created and attached to the VM.  
62 Values can be: standard, io1, cp1, cpio1, wig-io1, wig-cp1, wig-cpio1  
63 Default value: 'io1'  
64  
65 **.PARAMETER** VMName  
66 'VMName' is the hostname of the VM that will be created.  
67  
68 **.EXAMPLE**  
69 Example using default values:  
70 create-vm-with-volume -OpenStackProject "IT Windows Terminal Service Dev" -Hostgroup  
↪ "windows\_dev/ts" -AvailabilityZone "cern-geneva-c" -CMFMemberships "1059,1304"  
↪ -VolumeSize "1GB" -VMName "MyVM"

---

```

71
72 .EXAMPLE
73 Example with all parameters:
74 create-vm-with-volume -OpenStackProject "IT Windows Terminal Service Dev" -Hostgroup
75 ↪ "windows_dev/ts" -Environment "qa" -NovaImage "Windows 2016 Standard" -NovaFlavor
76 ↪ "m2.large" -LANDBResponsible "SUPPORT-WINDOWS-SERVERS" -AvailabilityZone
77 ↪ "cern-geneva-c" -CMFMemberships "1059,1304" -VolumeSize "1GB" -VolumeType "standard"
78 ↪ -VMName "MyVM"
79 #>
80 workflow create-vm-with-volume {
81     param(
82         [Parameter(Mandatory = $False)]
83         [string] $OpenStackProject = "IT Windows Terminal Service",
84
85         [Parameter(Mandatory = $True)]
86         [string] $Hostgroup,
87
88         [Parameter(Mandatory = $False)]
89         [string] $Environment = "production",
90
91         [Parameter(Mandatory = $False)]
92         [string] $NovaImage = "Windows 2016 Standard",
93
94         [Parameter(Mandatory = $False)]
95         [string] $NovaFlavor = "m2.xlarge",
96
97         [Parameter(Mandatory = $False)]
98         [string] $LANDBResponsible = "SUPPORT-WINDOWS-SERVERS",
99
100        [Parameter(Mandatory = $False)]
101        [string] $LANDBMainUser,
102
103        [Parameter(Mandatory = $False)]
104        [string] $AvailabilityZone = "cern-geneva-a",
105
106        [Parameter(Mandatory = $False)]
107        [string] $CMFMemberships = "1059",
108
109        [Parameter(Mandatory = $False)]
110        [string] $VolumeSize = "100GB",
111
112        [Parameter(Mandatory = $False)]
113        [string] $VolumeType = "io1",
114
115        [Parameter(Mandatory = $True)]
116        [string] $VMName
117    )
118    # preference variabbles for troubleshooting

```

```
117     $VerbosePreference = "Continue"
118     $DebugPreference = "Continue"
119     $WarningPreference = "Continue"
120     $ErrorActionPreference = "Continue"
121
122     $Conn = Get-AutomationConnection -Name 'adminconnection'
123     $Cred = Get-AutomationPSCredential -Name 'adminuser'
124
125     # Manipulating input parameters
126     $OpenStackProject = "'$OpenStackProject'" # enquoting OpenStackProject variable
127     $NovaImage = "'$NovaImage'" # enquoting NovaImage variable
128     $VolumeSettings = $VolumeSize + ":delete-on-terminate:type=" + $VolumeType
129     $VolumeLabel = "DATA"
130
131     $AIBSVMExitCodes = "EXIT CODES
132         0      All operations executed successfully.
133         2      Bad command line.
134         3      Bad user environment (no OpenStack's openrc.sh has been sourced)
135         4      Kerberos TGT not-existent or expired.
136         5      FQDN is invalid.
137         6      Userdata generation failed.
138         7      Userdata dump failed.
139         10     Foreman registration failed.
140         20     Host staging failed.
141         30     Nova boot failed.
142         40     Cinder volume operation failed.
143         50     Openstack authorization error"
144
145     if (!$CMFMemberships) {
146         Write-Verbose "Composing command to create VM: `r`n eval `$(ai-rc
            ↳ $OpenStackProject); ai-bs-vm --foreman-hostgroup $Hostgroup
            ↳ --foreman-environment $Environment --nova-image $NovaImage --nova-flavor
            ↳ $NovaFlavor --landb-responsible=$LANDBResponsible
            ↳ --landb-mainuser=$LANDBMainUser --nova-availabilityzone $AvailabilityZone
            ↳ --nova-attach-new-volume disk1=$VolumeSettings $VMName"
147
148         $vmCreateCmd = [Scriptblock]::Create("eval `$(ai-rc $OpenStackProject); ai-bs-vm
            ↳ --foreman-hostgroup $Hostgroup --foreman-environment $Environment
            ↳ --nova-image $NovaImage --nova-flavor $NovaFlavor
            ↳ --landb-responsible=$LANDBResponsible --landb-mainuser=$LANDBMainUser
            ↳ --nova-availabilityzone $AvailabilityZone --nova-attach-new-volume
            ↳ disk1=$VolumeSettings $VMName")
149     }
150     else {
151         # Using CMF memberships
```

---

```

152 Write-Verbose "Composing command to create VM: `r`n eval `$(ai-rc
    ↳ $OpenStackProject); mkdir -p cmf_memberships_dir; echo
    ↳ `'$'#cmf\n[NSC]\nMembership=$CMFMemberships' >
    ↳ cmf_memberships_dir/cmf_memberships.txt; ai-bs-vm --foreman-hostgroup
    ↳ $Hostgroup --foreman-environment $Environment --nova-image $NovaImage
    ↳ --nova-flavor $NovaFlavor --landb-responsible=$LANDBResponsible
    ↳ --landb-mainuser=$LANDBMainUser --nova-availabilityzone $AvailabilityZone
    ↳ --nova-attach-new-volume disk1=$VolumeSettings --userdata-dir
    ↳ cmf_memberships_dir $VMName"

153
154 $vmCreateCmd = [Scriptblock]::Create("eval `$(ai-rc $OpenStackProject); mkdir -p
    ↳ cmf_memberships_dir; echo `'$'#cmf\n[NSC]\nMembership=$CMFMemberships' >
    ↳ cmf_memberships_dir/cmf_memberships.txt; ai-bs-vm --foreman-hostgroup
    ↳ $Hostgroup --foreman-environment $Environment --nova-image $NovaImage
    ↳ --nova-flavor $NovaFlavor --landb-responsible=$LANDBResponsible
    ↳ --landb-mainuser=$LANDBMainUser --nova-availabilityzone $AvailabilityZone
    ↳ --nova-attach-new-volume disk1=$VolumeSettings --userdata-dir
    ↳ cmf_memberships_dir $VMName")

155 }
156
157 $StartWorkflowTime = Get-Date -Format G
158 Write-Output "$StartWorkflowTime - Creating VM."
159
160 $vmCreateCmdResult = Invoke-SSHCommand -Connection $Conn -ScriptBlock $vmCreateCmd
161 $VolumeID = GetVolumeIDFromErrorMsg -ErrorMsg $vmCreateCmdResult.Error
162
163 Write-Verbose "Command executed: $($vmCreateCmdResult.CommandText)"
164
165 if ($vmCreateCmdResult.ExitStatus -ne 0) {
166     Write-Output "$(Get-Date -Format G) - It was not possible to create the VM. The
        ↳ Error: `r`n $($vmCreateCmdResult.Error) `r`n ExitCode:
        ↳ $($vmCreateCmdResult.ExitStatus) `r`n Check the ai-bs-vm Exit Codes by runing
        ↳ 'man ai-bs-vm' on a aiadm machine."
167     Write-Error "It was not possible to create the VM. The Error: `r`n
        ↳ $($vmCreateCmdResult.Error) `r`n ExitCode: $($vmCreateCmdResult.ExitStatus)
        ↳ `r`n Check the ai-bs-vm Exit Codes by runing 'man ai-bs-vm' on a aiadm
        ↳ machine."
168     if ($VolumeID) {
169         Write-Output "$(Get-Date -Format G) - Deleting volume $VolumeID"
170         $deleteVolResult = DeleteVolume -VolumeID $VolumeID -Connection $Conn
            ↳ -OpenStackProject $OpenStackProject
171         Write-Verbose "delete volume result:`r`n $($deleteVolResult)"
172     }
173     Write-Output "$AIBSVMExitCodes"
174     Write-Output "$(Get-Date -Format G) - Finished with errors. VM not created."
175     exit
176 }
177 else {
178     Write-Verbose "Create VM command output `r`n $($vmCreateCmdResult.Error)"
179 }

```

```
180 Write-Verbose "VM Created. Now doing a Checkpoint-Workflow."
181
182 Checkpoint-Workflow
183
184 Write-Output "$(Get-Date -Format G) - Waiting for VM to be up and running."
185 try {
186     $totalWaitTime = WaitForVM -SleepInterval 300 -VMName $VMName -Connection $Conn
187     ↪ -OpenStackProject $OpenStackProject # Sleep for 300 secs = 5 min, until VM is
188     ↪ ready to receive commands
189 }
190 catch {
191     Write-Output "$(Get-Date -Format G) - an error occured when creating the VM on
192     ↪ OpenStack."
193     Write-Output "$(Get-Date -Format G) - Deleting VM $VMName"
194     $deleteVMResult = DeleteVm -VMName $VMName -Connection $Conn -OpenStackProject
195     ↪ $OpenStackProject
196     Write-Verbose "Delete VM result:`r`n $($deleteVMResult)"
197
198     Write-Output "$(Get-Date -Format G) - Deleting volume $VolumeID"
199     $deleteVolResult = DeleteVolume -VolumeID $VolumeID -Connection $Conn
200     ↪ -OpenStackProject $OpenStackProject
201     Write-Verbose "Delete volume result:`r`n $($deleteVolResult)"
202
203     Write-Output "$(Get-Date -Format G) - Finished with errors. VM not created."
204     exit
205 }
206
207 Write-Output "$(Get-Date -Format G) - VM is ready. Now waiting for CMF to install all
208 ↪ the packages."
209 $totalWaitTimeCMF = WaitForCMF -SleepInterval 180 -VMName $VMName
210 Write-Verbose "CMF is ready. Now doing a Checkpoint-Workflow."
211
212 Checkpoint-Workflow
213
214 Write-Output "$(Get-Date -Format G) - Rebooting VM."
215 InlineScript {
216     Restart-Computer -Wait -For WinRM -ComputerName $Using:VMName -Force
217 }
218
219 Write-Verbose "VM restarted. Now doing a Checkpoint-Workflow."
220 Checkpoint-Workflow
221
222 Write-Output "$(Get-Date -Format G) - Formatting volume."
223 # Format the attached volume.
224 InlineScript
225 {
226     Write-Verbose "Invoking command to Initialize, Create and Format the new Disk"
```

---

```

221 Write-Verbose "Formatting Volume using PowerShell code: Get-Disk | Where-Object
↪ partitionstyle -eq 'raw' | Initialize-Disk -PartitionStyle MBR -PassThru |
↪ New-Partition -AssignDriveLetter -UseMaximumSize | Format-Volume -FileSystem
↪ NTFS -NewFileSystemLabel $Using:VolumeLabel -Confirm:$false"
222
223 $formatDiskResult = Get-Disk |
224     Where-Object partitionstyle -eq 'raw' |
225     Initialize-Disk -PartitionStyle MBR -PassThru |
226     New-Partition -AssignDriveLetter -UseMaximumSize |
227     Format-Volume -FileSystem NTFS -NewFileSystemLabel $Using:VolumeLabel
        ↪ -Confirm:$false
228 } -PSComputerName $VMName
229
230
231 Checkpoint-Workflow
232
233 # Get all puppet errors from the event log
234 $puppetErrors = Get-EventLog -LogName "Application" -Source "Puppet" -EntryType
↪ "Error" -PSComputerName $VMName -After $StartWorkflowTime
235
236 # Filter errors: get only certificate related errors
237 $puppetCertErrors = $puppetErrors | Where-Object {$_.Message -like "*certificate*"
↪ -or $_.Message -like "*SSL*" -or $_.Message -like "*503*"}
238
239 # Count the number of certificate related errors to see if there are any
240 $puppetCertErrorsCount = ($puppetCertErrors | Measure-Object).Count
241 Write-Verbose "total puppet cert errors: $puppetCertErrorsCount"
242
243 if ($puppetCertErrorsCount -gt 0) {
244     Write-Output "$(Get-Date -Format G) - Fixing puppet certificate on $VMName."
245     FixPuppetCert -ComputerName $VMName -Credential $Cred
246
247     Write-Output "$(Get-Date -Format G) - Rebooting VM."
248     InlineScript {
249         Restart-Computer -Wait -For WinRM -ComputerName $Using:VMName -Force
250     }
251
252     Write-Verbose "VM restarted. Now doing a Checkpoint-Workflow."
253     Checkpoint-Workflow
254 }
255
256 Write-Output "$(Get-Date -Format G) - Restart Puppet Agent Service."
257 InlineScript
258 {
259     Write-Verbose "Check puppet service status: `r`n $((Get-Service puppet).Status)"
260
261     Write-Verbose "Stopping puppet service"
262     Stop-Service puppet
263     Write-Verbose "Check puppet service status: `r`n $((Get-Service puppet).Status)"
264

```

```
265     Write-Verbose "Starting puppet service"
266     Start-Service puppet
267     Write-Verbose "Check puppet service status: `r`n $((Get-Service puppet).Status)"
268
269 } -PSComputerName $VMName
270
271 $EndWorkflowTime = Get-Date -Format G
272 $TotalWorkflowTime = New-TimeSpan -Start $StartWorkflowTime -End $EndWorkflowTime
273 Write-Output "$EndWorkflowTime - Finished. VM created with an attached volume. Total
↳ time: $TotalWorkflowTime"
274
275 # Send e-mail
276 $notifyAdmins = Get-AutomationVariable -Name "NotifyAdmins"
277
278 Send-Notification -To $notifyAdmins -Subject "VM $VMName created" -Body "VMName:
↳ $VMName `r`n Image: $NovaImage `r`n Flavor: $NovaFlavor `r`n Foreman Hostgroup:
↳ $Hostgroup `r`n Environment: $Environment `r`n LANDB responsible:
↳ $LANDBResponsible `r`n Availability zone: $AvailabilityZone `r`n CMF Memberships:
↳ $CMFMemberships `r`n Volume attached size: $VolumeSize `r`n Volume attached type:
↳ $VolumeType `r`n Total time spent: $TotalWorkflowTime.`r`n`r`n Puppet Agent will
↳ start executing and soon the first report will appear on Foreman (max. 60 min.)."
279
280 #####
281 ##                               Workflow code finished                               ##
282 ##                               Auxiliary Functions below                               ##
283 #####
284
285
286 <#
287 .SYNOPSIS
288 GetVolumeIDFromErrorMsg function finds the VolumeID that comes in the output of the
↳ VM creation command.
289 In this output there are a lot of lines, so it needs to be parsed to get the VolumeID
↳ created by the command.
290
291 .DESCRIPTION
292 Parse the output error message from the command to create a VM ('ai-bs-vm') and
↳ retrieve the VolumeID.
293
294 .PARAMETER ErrorMsg
295 The error message string that the VM create command produces.
296 .EXAMPLE
297 $VolumeID = GetVolumeIDFromErrorMsg -ErrorMsg $vmCreateCmdResult.Error
298 #>
299 function GetVolumeIDFromErrorMsg ([string] $ErrorMsg) {
300     $separator = @"(Getting volume", "from Cinder)"
301     $resultStrings = $ErrorMsg.Split($separator,
↳ [System.StringSplitOptions]::RemoveEmptyEntries)
302     if ($resultStrings.Count -gt 1) {
303         $VolumeIDstr = $resultStrings[1]
```



---

```
304         $VolumeID = $VolumeIDstr.Trim(' ', '"')
305         return $VolumeID
306     }
307     else {
308         return $null
309     }
310 }
311
312 }
```

---